THE UNIVERSITY *of* EDINBURGH

J-model: an Open and Social Ensemble Learning Architecture for Classification

Jinhan Kim

Doctor of Philosophy

Centre for Intelligent Systems and their Applications
School of Informatics
University of Edinburgh

2012

Abstract

Ensemble learning is a promising direction of research in machine learning, in which an ensemble classifier gives better predictive and more robust performance for classification problems by combining other learners. Meanwhile agent-based systems provide frameworks to share knowledge from multiple agents in an open context. This thesis combines multi-agent knowledge sharing with ensemble methods to produce a new style of learning system for open environments.

We now are surrounded by many *smart objects* such as wireless sensors, ambient communication devices, mobile medical devices and even information supplied via other humans. When we coordinate smart objects properly, we can produce a form of collective intelligence from their collaboration. Traditional ensemble methods and agent-based systems have complementary advantages and disadvantages in this context. Traditional ensemble methods show better classification performance, while agent-based systems might not guarantee their performance for classification. Traditional ensemble methods work as closed and centralised systems (so they cannot handle classifiers in an open context), while agent-based systems are natural vehicles for classifiers in an open context.

We designed an open and social ensemble learning architecture, named J-model, to merge the conflicting benefits of the two research domains. The J-model architecture is based on a service choreography approach for coordinating classifiers. Coordination protocols are defined by interaction models that describe how classifiers will interact with one another in a peer-to-peer manner. The peer ranking algorithm recommends more appropriate classifiers to participate in an interaction model to boost the success rate of results of their interactions. Coordinated participant classifiers who are recommended by the peer ranking algorithm become an ensemble classifier within J-model.

We evaluated J-model's classification performance with 13 UCI machine learning benchmark data sets and a virtual screening problem as a realistic classification problem. J-model showed better performance of accuracy, for 9 benchmark sets out of 13 data sets, than 8 other representative traditional ensemble methods. J-model gave better results of specificity for 7 benchmark sets. In the virtual screening problem, J-model gave better results for 12 out of 16 bioassays than already published results. We defined different interaction models for each specific classification task and the peer ranking algorithm was used across all the interaction models.

Our research contributions to knowledge are as follows. First, we showed that service choreography can be an effective ensemble coordination method for classifiers in an open context.

Second, we used interaction models that implement task specific coordinations of classifiers to solve a variety of representative classification problems. Third, we designed the peer ranking algorithm which is generally and independently applicable to the task of recommending appropriate member classifiers from a classifier pool based on an open pool of interaction models and classifiers.

Acknowledgements

I pay my tribute of praise to my wife[1]. She gave me everything of her. Thank you.

Father[2], I love you.

Thank you all for your friendship and discussion with me. They are Xi[3], Shariar[4], Gaya[5], Tommy[6], Omar[7] and Siu-wai[8].

To my family, thank you for your support. They are my mother[9], my mother-in-law[10] and my son[11].

Paolo[12] is my friend and the secondary supervisor. Thank you and see you after the restoration of your health.

Dear Dave[13], I always say “really thank you”. You are one of the best and the wisest men whom I have ever met. You have guided me very well and given me invaluable *knowledge* and wisdom. You yourself have stood as my *role* model.

[1] Youn Jin Choi
[2] Choong Kyu Kim
[3] Xi Bai
[4] Shahriar Bijani
[5] Gayathri Nadarajan
[6] Thomas French
[7] Omar Montano Rivas
[8] Siu-wai Leung
[9] Chun Ja Jang
[10] Book Hee Nam
[11] Jaeyoun Kim
[12] Paolo Besana
[13] Dave Robertson, my primary supervisor

Declaration

I declare that this thesis was composed by myself, that the work contained herein is my own except where explicitly stated otherwise in the text, and that this work has not been submitted for any other degree or professional qualification except as specified.

(*Jinhan Kim*)

Contents

List of Figures

List of Tables

Chapter 1

Introduction

1.1 Motivation

Machine learning [65], as studied in this thesis, is performed by algorithms that automatically build a trained pattern from observed data. A trained pattern is a generalised rule that is able to label new or unobserved data. Machine learning helps us avoid having to extract patterns manually from data and beyond that it can suggest patterns that we could not consider using human abilities alone.

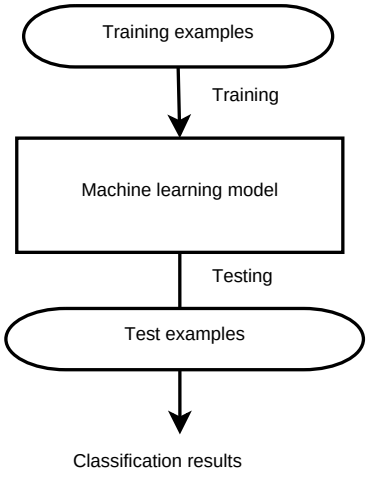

Figure 1.1: Machine learning process. A machine learning model trains training examples (observed data) and the training process automatically makes a pattern in the model. For test examples (unobserved data), the machine learning model can suggest class labels for each example based on the constructed pattern.

In the history of development of machine learning, ensemble methods [70, 82] are a notable direction driven by the need to incorporate diversity in learning systems. Ensemble methods are intended to produce better and more robust prediction performance through combining multiple machine learning models. Using a divide-and-conquer applicable approach for training over large data sets, ensemble methods can manage large data better than single machine learning models.

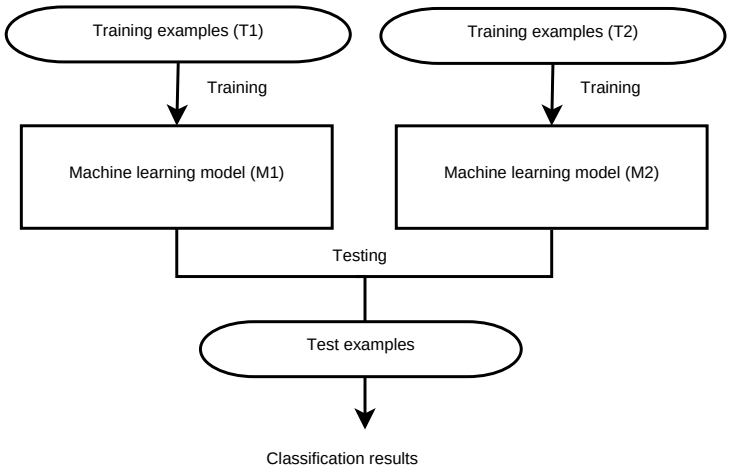

Figure 1.2: Illustration of the ensemble learning process ensemble learning process. The machine learning models M1 and M2 train different training examples T1 and T2 respectively. Predictions from their suggested patterns are combined for testing test examples.

Ensemble methods have successfully been applied to several application domains. For example, remote sensing, person recognition and medical applications are their well-applicable problem domains [71].

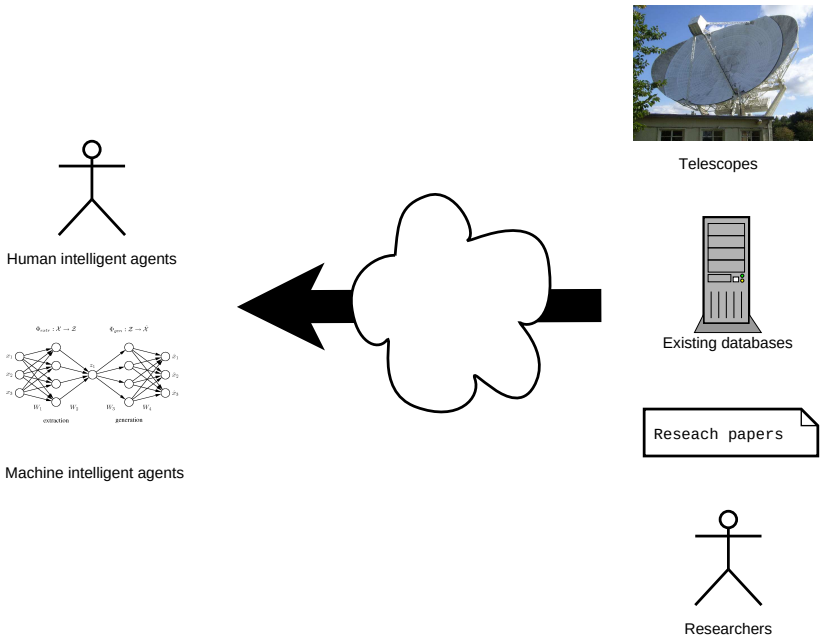

Figure 1.3: Astronomy research

Figure 1.3 illustrates what happens in Astronomy research [8]. Intelligent agents that are humans or machines[1] collect Astronomy data and information from several sources. Sources can be telescopes[2] that generates astronomical data, data-warehouses, research papers and other research colleagues. Intelligent agents compile data and information and extract knowledge from this process. Their knowledge is shared and provided for development of Astronomy research.

Let us examine the dynamics of the Astronomy research from an agent viewpoint. Agents have the following properties.

- They are diverse. They research their own interests with different backgrounds. There can be conflicts between their opinions when analysing and interpreting data.
- An agent can change its interest. At one point, an agent may be focusing on collecting data from telescopes. Later it may be writing a paper using existing databases. A new agent can join the research community and some may retire from the community.
- They are distributed. They are scattered all over the world.
- They get together or split apart depending on their research purpose.

[1]For example, an artificial neural network - http://phdthesis-bioinformatics-maxplanckinstitute-molecularplantphys.matthias-scholz.de/fig_NLPCAs_bottleneck_autoassociative_autoencoder_neural_network_thumb200.png

[2]The Mark II Telescope - http://s0.geograph.org.uk/photos/03/67/036740_fa63d32e.jpg

They are in an open dynamic system, not a closed system. In open dynamic systems, different agents interact with one another autonomously within a framework of social conventions and freely join or leave the community. In contrast, in closed dynamic systems, agents' interactions are hard wired and a system is closed for other agents to join in the system.

When we need to get more valuable and integrated knowledge about Astronomy, we can ask advice to several experts on Astronomy (the intelligent agents). We can try to apply ensemble methods for our purpose because ensemble methods provide verified frameworks to combine opinions from multiple experts as we introduced above.

We, however, encounter a critical problem. It is that ensemble methods have been successful under the closed dynamics of classifiers or trained models. An ensemble method generates base classifiers and combines their predictions for test examples. In this process, there are not any interactions among base classifiers and the classifiers are fixed. In other words, traditional ensemble methods work under this assumption and are not well adapted to open dynamics problems.

We have another strong research heritage that handles multiple agents to get intelligence from their collaboration in the artificial intelligence area. Marvin Minsky's research on *society of mind* [64] is the beginning. Multi-agent systems (MAS) [33] are for building distributed systems. Their computational components called *agents* are autonomous to control their behaviour for their own goals. Agents interact with one another. Internet environments such as service oriented architectures (SOAs) [30] provide architectural bases for coordinating distributed and interoperable agents. These systems suggest appropriate concept and frameworks in which we can handle an open dynamics of multiple agents to get some aspects of global intelligence from agents.

Agent-based systems also have a problem that is tricky to solve. Agent-based systems in an open environment in which agents have autonomy are difficult to control in order to give better prediction performance than ensemble methods can provide. A main reason for this is that it is challenging to select the best agents and coordinate them for prediction among agents in an open environment. Ensemble methods work in a closed environment (base classifiers have a closed dynamics). Ensemble methods generate member classifiers and combine them for prediction under tight control. Thus the advantages which ensemble methods and agent-based systems give are complementary for prediction under an open environment, but only if a means to combine them can be found.

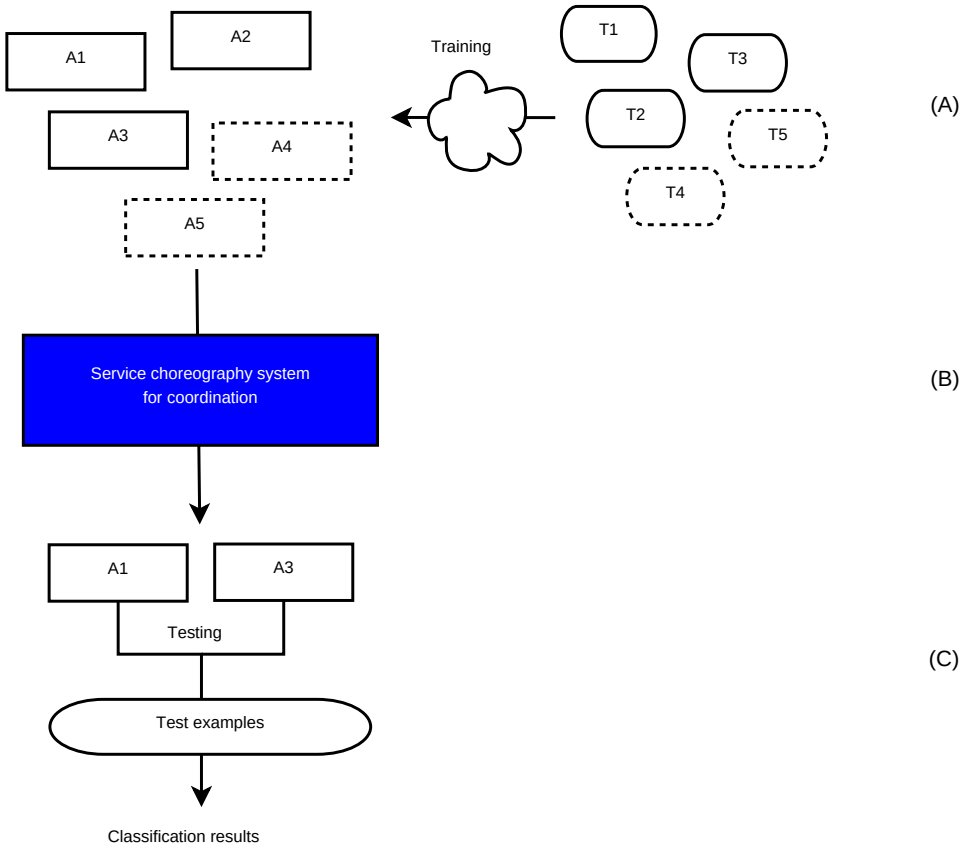

Figure 1.4: Service choreography system for coordinating agents. T is a training data and A is an agent. Training data and agents in dotted boxes mean that they are absent at this point.

Figure 1.4 shows our proposed approach to solving the problems that ensemble methods and agent-based systems have for prediction. This is also a proposed way to merge the advantages of both sides. In the figure, part (A) is an environment in which classifier agents show their open dynamics. We offered Figure 1.3 as an example of this open environment. Part (B) is a system that implements our approach. It takes classifier agents as inputs and gives an ensemble classifier as an output result. It selects and coordinates appropriate agents for prediction. Coordinated agents by the system become an ensemble classifier. The part (C) is an ensemble classifier of agents. We explain the architecture of the system in Chapter 4.

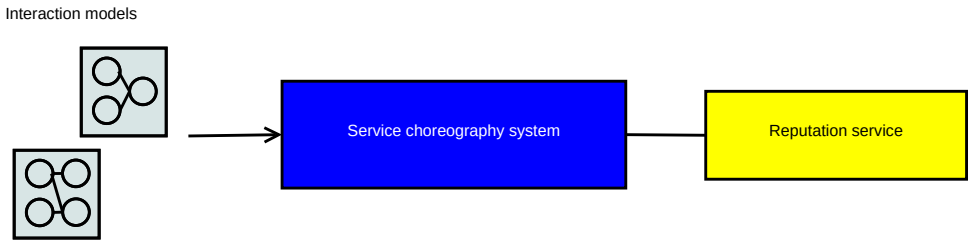

Figure 1.5: Service choreography system with interaction models and reputation service

A service choreography system [74] is a structure to support the delivery process from open dynamic classifier agents to ensemble classifiers. This is a solution for handling multiple classifiers in an open environment. Figure 1.5 summarises the top-level operational components of our approach. There are two components: interaction models and a reputation service for a service choreography system. Interaction models are workflows designed for classification tasks. Agents take roles in an interaction model and collaborate to provide predictions. An individual interaction model filled with classifier agents becomes an ensemble classifier. The reputation service recommends appropriate agents for prediction to a service choreography system. These task-oriented interaction models and the reputation service consequentially boost the prediction performance of an ensemble classifier. We explain the reputation service in Chapter 5 and interaction models for classification in Chapter 6 respectively.

1.2 Objectives

The objectives of this thesis can be summarised by the following two aims:

- To suggest an ensemble learning architecture that can coordinate classifiers in an open context.
- To suggest a reputation mechanism that can recommend appropriate classifiers in an open context for better classification performance.

The first objective will be realised by reconstructing ensemble learning based on a service choreography architecture. The second objective includes two sub objectives. One is that the reputation mechanism should be generally applied to various coordination models. The other is that the mechanism should be robust to recommend for services in an open context.

1.3 Research hypothesis

Our suggested ensemble learning architecture based on a service choreography paradigm is named J-model. J-model delivers a comparative advantage that it is a more appropriate working architecture for an open classifier environment than previous traditional ensemble learning methods. In this context, we need to evaluate a prediction performance of J-model.

Research hypotheses which we set in this thesis are

1. J-model's prediction performance approaches the performance of traditional ensemble methods.
2. J-model's prediction performance approaches the performance of traditional ensemble methods in practical time.
3. J-model is applicable to realistic learning problems.
4. Minimal parameterisation is required for J-model prediction.

1.4 Thesis structure

Chapter 2 Background In Chapter 2, we introduce 5 research areas. First, we define properties that services have in an open context. Second, we explain what ensemble methods are and their advantages as machine learning algorithms. Third, we visit a service-oriented architecture focusing on orchestration and choreography coordination approaches. Fourth, we give an explanation about OpenKnowledge framework. General interaction models described with the lightweight coordination calculus are an essential part of OpenKnowledge framework. Last, we survey social reputation that has features of network effects and power-law distributions.

Chapter 3 Architectural migration The main purpose of this chapter is to provide a smooth architectural migration from traditional ensemble learning systems to an open and social ensemble learning architecture. The chapter is composed of three different sub migrations. In Section 3.2, we migrate from classifiers in traditional ensemble methods to Web services that have the ability of classification. In Section 3.3, we introduce service choreography based coordination instead of classifier aggregation. In Section 3.4, we show how to get better prediction performance for open and social ensemble learning.

Chapter 4 J-model architecture We call our open and social ensemble learning architecture J-model. In this chapter, we organise migrated parts shown in Chapter 3 into J-model and explain the J-model architecture. J-model's operation is divided into three phases of discovery, enactment and analysis. We explain the three phases of J-model in Section 4.3. J-model's operation also is analysed based on the traditional machine learning process of training and test in Section 4.4.

Chapter 5 Peer ranking service We explain a reputation service called the peer ranking service. In Section 5.2, we show what the peer ranking service does to recommend more appropriate classification services. In Section 5.3, we define the peer ranking algorithm. After that, we give examples of peer ranking recommendation under both a static classifier condition and a dynamic classifier condition. Last, we provides two sorts of termination conditions on service recommendation - the number of interactions and a performance metric.

Chapter 6 Interaction models for classification We firstly suggest interaction models in which participating classifiers are fixed in design-time (closed interaction models). Next, we show 6 examples of interaction models in which participating classifiers are not fixed in design-time but are chosen at run-time (open interaction models). Open interaction models are for classification services in an open context. Each example is designed to achieve a different classification task.

Chapter 7 Experiments The experiment chapter is composed of four independent experiments. In Section 7.3, we verify that more appropriate peers (classification services) for classification are separated from the broader peer pool by the peer ranking algorithm. In Section 7.4, we confirm that separated peers converge to common optima. For this, we set a different initial selection of peers repeatedly. Learning curves in Section 7.5 give a connection between chosen peers by the peer ranking algorithm and their classification performance. Learning curves show changes of performance of chosen peers over interactions. In Section 7.6, we measure J-model's classification performance with standard machine learning benchmark data sets and compare its performance with other representative traditional ensemble methods. In the last section of 7.7, we apply J-model to a realistic classification problem: virtual screening.

Chapter 8 Discussion We discuss a balance and a bias between exploration and exploitation on peers for the results of the peer separation. We explain why there are big falls and cyclic patterns in learning curves. For the virtual screening results, we pay attention to the quality of a peer pool. Finally we discuss appropriate ensemble sizes and parameterisation of J-model.

Chapter 9 Related work We expand research described in the background chapter and our J-model research to the relevant research boundaries focusing on the topic of collaborative learning in an open context. The topic includes distributed ensemble classification, agent-based

distributed data mining, multi-agent learning, open multi-agent systems, distributed workflows, ensemble selection and social recommendation systems.

Chapter 10 Conclusions We check whether the hypothesis of this thesis is verified. We summarise our contributions to knowledge from this work. We suggest future work as extensions of the work which had done in this thesis.

Chapter 2

Background

2.1 Services in open context

We categorise services in open context according to one or more of the following features:

- Unspecific

  Services are independently implemented by different designers. This entails that services may give variable behaviours. The behaviours might conflict with one another for individual goals. Services can be buggy or even malicious.

- Join, leave and change

  Services easily join or leave a system at their will at runtime. They temporarily take or release their roles. They obtain or lose resources.

- Distributed

  Services are located on a distributed network, not only at a single location.

- Shared

  Different service applications can share some of the same services. Services may be reused.

2.2 Ensemble methods

2.2.1 Concept

Theorem 2.1. Condorcet's jury theorem
Each voter has a probability p of being correct. If the probability of a majority of voters being correct is M then:
$p > 0.5$ implies $M > p$. Also M approaches 1, for all $p > 0.5$ as the number of voters approaches infinity.

Here the votes are independent and there are only two possible outcomes.

Theorem 2.1 [24] supports that a correct decision probably can be obtained by simply combining the votes of a large enough jury that is composed of voters whose judgements are slightly better than a random vote. The theorem is a theoretical basis for ensemble learning.

2.2.2 Algorithms

The first step to build an ensemble classifier is to generate multiple trained models. The models are base classifiers. The training data upon which each base classifier trains is a differently manipulated data set from the original data set. The second step is to combine the base classifiers. Their predictions are aggregated by an unweighted or weighted vote for a final prediction.

Algorithm 2.1 Bagging algorithm

Require: I (a base inducer), T (number of iterations), S (the original training set), μ (the sample size)

1: $t \leftarrow 1$
2: **repeat**
3: $S_t \leftarrow$ a sample of μ instances from S with replacement.
4: Construct classifier M_t using I with S_t as the training set
5: $t \leftarrow t + 1$
6: **until** $t > T$

Bagging (bootstrap aggregating) [11] in Algorithm 2.1[1] is one of the most well-known ensemble methods. In the Bagging algorithm, each member classifier is constructed from a dif-

[1] From [83]

ferent training data set. Each training data set is generated by sampling from an original data set with uniformly random replacement.

Algorithm 2.2 Boosting algorithm

Require: I (a weak inducer), S (training set) and k (the sample size for the first classifier)

Ensure: M_1, M_2, M_3

1: $S_1 \leftarrow$ Randomly selected $k < m$ instances from S without replacement;

2: $M_1 \leftarrow I(S_1)$

3: $S_2 \leftarrow$ Randomly selected instances (without replacement) from $S - S_1$ such that half of them are correctly classified by M_1.

4: $M_2 \leftarrow I(S_2)$

5: $S_3 \leftarrow$ any instances in $S - S_1 - S_2$ that are classified differently by M_1 and M_2.

6: $M_3 \leftarrow I(S_3)$

Boosting [35] in Algorithm 2.2[2] is another mostly well-known ensemble method. Similar to bagging, base classifiers are generated by resampling a training data set. Boosting, however, uses a different resampling mechanism. The mechanism samples a training data set for a base classifier to train incorrectly classified samples of the previous iteration. Boosting boosts prediction performance of base classifiers through making them be more sensitive at incorrectly classified instances.

2.2.3 Advantages

The advantages of ensemble methods can be summarised as follows:

- Classification performance

  Ensemble methods should obtain better predictive performance with individual predictions combined appropriately. Each base classifier covers a different local search space. The base classifiers (hypotheses) appear equally accurate. Combining them gets a good approximation of the unknown true hypothesis.

- Robustness

  Covering different local search spaces helps ensemble learning to be robust for classification. It reduces the likelihood of an unfortunate selection of a poor classifier. It also lessens the overfitting problem.

[2] From [83]

- Flexibility

  Ensemble methods adaptively train to different classification problems. For example, boosting increasingly exposes incorrectly classified instances to each next base classifier in its iteration process. Ensemble methods also can use any sort of base classifier and any combination of algorithms. These features makes ensemble learning more adaptable to various classification applications.

- Too much and too little data

  Ensemble methods can be used for learning from large data. A single machine learner cannot learn a large data easily. For ensemble learning, a large data set can be divided into smaller data sets. Each data set is used to train a classifier in ensemble learning. The random subspace method [45] is a representative algorithm using this approach.

  Ensemble methods can learn too little data. For example, bootstrapping in bagging method resamples data with replacement. The bootstrapped data acquires independent sample distribution. A base classifier learns a different bootstrapped data. This brings a more general learning pattern to a final classifier.

2.3 Service-oriented architecture

Having described the basics of ensemble learning, we now switch attention to the environment assumed by this thesis - a service oriented architecture.

2.3.1 Services

A service is a software component that encapsulates functions and data. A service is designed for a business functionality. Services are loosely coupled. They are accessible over a network. They are combined and reused for different service applications. Client services communicate with server services through interface by passing data in a well-defined and shared format.

2.3.2 Workflows

Multiple services are coordinated for a shared goal. A workflow is a dynamic set of activities in which services participate and interact to achieve a shared goal. An executable workflow is

a committed coordination of services.

2.3.3 Orchestration and choreography coordinations

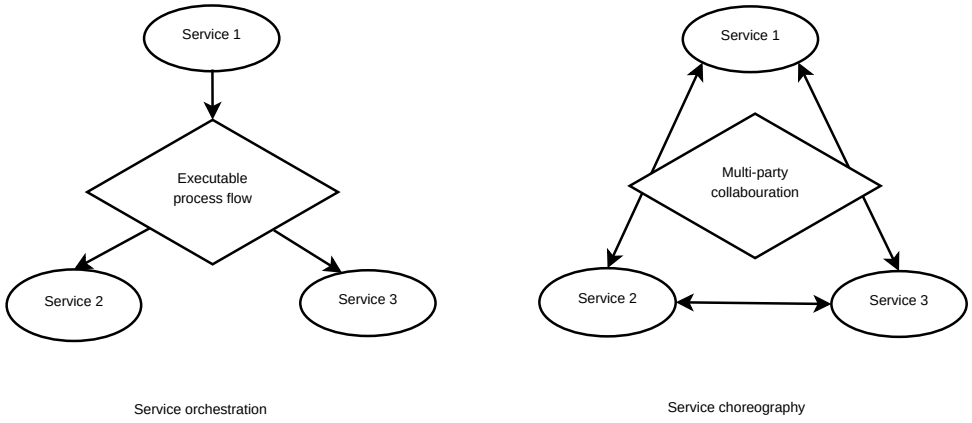

Figure 2.1: Service orchestration and service choreography. Service orchestration executes a process flow. Services in orchestration are subroutines on the flow. Service choreography declares roles that services will take. A multi-party collaboration occurs among participating services.

2.3.3.1 Service orchestration

Service orchestration [74] is the process of executing a coordination process for services based on a central means of coordination.

Execution by a single orchestrator A single orchestrator specifies an execution process of services. The orchestrator defines which service takes what sub process. Services in orchestration are passive for coordination (they do not need to have an understanding of the broader coordination process).

Orchestration languages The Business Process Execution Language (WS-BPEL[3]) [73] is a representative orchestration language. The Yet Another Workflow Language (YAWL[4]) [95] and the XML Process Definition Language (XPDL[5]) [87] are orchestration languages. These are imperative paradigm languages.

[3]http://docs.oasis-open.org/wsbpel/2.0/wsbpel-v2.0.html
[4]http://www.yawlfoundation.org/
[5]http://www.wfmc.org/xpdl.html

2.3.3.2 Service choreography

Service choreographies [74] describe required observable interactions between services from an external viewpoint.

Enactment Service choreographies are not executed. They are enacted. Services adopt and perform the roles which are described in an interaction protocol. Coordination happens from their execution of roles.

A global perspective The meaning of a global perspective in choreography coordination is that each participating service knows its role in coordination and they collaborate for coordination. There is no central engine leading their coordination.

Choreography languages The most well-known choreography languages are Web Services the Choreography Description Language (WS-CDL[6]) [49], the Web Service Choreography Interface (WSCI[7]) [3] and the Ontology Web Language for Services (OWL-S[8]) [60]. The Lightweight Coordination Calculus (LCC) [78, 77] which J-model uses to describe its interaction models is also a choreography language.

2.3.3.3 Advantages of choreography

1. Choreography is able to adapt to changing and uncertain real-world classifiers by separating role definition from the choice of agents to participate in each role.

   The performances of the classifiers unpredictably change over time. They may even give errors. The choreography system just defines high-level protocols that describe what roles the classifiers take, what interactions they do and who they interact with. The actual classifiers are determined at run-time, not design-time. So the status of classifiers can be adaptively reflected in the choreography system.

2. Choreography may handle more complex situations such as additional classifiers participating and missing classifiers.

[6] http://www.w3.org/TR/ws-cdl-10/
[7] http://www.w3.org/TR/wsci/
[8] http://www.w3.org/Submission/OWL-S/

When new classifiers are introduced, the orchestration system needs to explicitly contain them in the system's centralised combining process to use them. When some classifiers are missed, the orchestration system will experience errors because the system's control flow cannot proceed on run-time. An adaptive choreography system, however, has resilience against these problems because other classifiers may take over the roles that the missing ones had have vacated.

3. Choreography may scale better to open systems.

   In open systems, agents are (arguably) less likely to subscribe as a collective to centralised orchestration but choreography offers a *take it or leave it* alternative to controlled coordination that respects the autonomy of the individual agent.

   A local prediction arises in each distributed classification service and a coordination of their predictive abilities builds a global predictive ability. Service orchestration systems preform this coordination work using a centralised controller. A centralised controller has to contain every information for coordination and manage all of the participating classifiers. This feature of service orchestration systems is not suitable for increasing number of classifiers. Meanwhile, in service choreography systems, a coordination of distributed classifiers occurs from an enactment of participating classifiers who take their roles defined in a interaction protocol. Classifiers who satisfy described roles work together to be collaborative. Information for a coordination and the classifier management also is also distributed at each classifier. This feature makes service choreography systems scalable to open systems.

2.4 OpenKnowledge framework

Web services are software components that are invoked through interaction protocols on the Web. Interaction protocols are described using formalised coordination languages. The OpenKnowledge framework [80] is a fully distributed choreography system based on a peer-to-peer technology. Users publish interaction protocols called *interaction models* on the OpenKnowledge system. Programmers design and register Web services on the system. The OK system provides a interaction-centred mechanism for sharing knowledge from services by sharing interaction models in a peer-to-peer environment.

2.4.1 Service, participant and peer

We need to explicitly explain what services, participants and peers are in different contexts. *Services* are software components on a network as we explained. *Participants* are services that take roles in workflows, coordination protocols or interaction models in OpenKnowledge. We call participants *peers* on a peer-to-peer system. The OpenKnowledge framework is a peer-to-peer system. Services on the OpenKnowledge framework are therefore called peers.

2.4.2 Interaction models

An interaction model is a coordination protocol describing knowledge for a specific task. It is specified using the Lightweight Coordination Calculus (LCC) language.

Definition 2.1. *Example of interaction model: researcher and omics_lab*[9]

```
a(researcher,A) ::
    null <- get_query(Query) then
    a(researcher(Query,IDs,Results),A) <- getPeers(omics_lab,IDs) then
    null <- visualise(Results)

a(researcher(Query,IDs,Results),C) ::
    (
    query(Query,ID) => a(omics_lab,_) <- IDs = [ID|RIDs] then
    answer(Result) <= a(omics_lab,_) <- Results = [Result|RR] then
    a(researcher(Query,RIDs,RR),C)
    )
    or
    null <- IDs = []

a(omics_lab,L) ::
    query(Query,ID) <= a(researcher,_) then
    answer(Result) => a(researcher,_) <- find_hit(Query,ID,Result) then
    a(omics_lab,L)
```

Definition 2.1 is an example of interaction model in the omics research of the OpenKnowledge project[10] [1]. There are two roles of `researcher` (`A`) and `omics_lab` (`L`). `researcher` (`C`) takes delegation from `A` for querying messages. `A` selects a query and gets omics labs to whom the query will be sent. `C` queries to omics labs and collects their answers. `L` receives a query from `C` and sends a hit answer back to `C`. `A` finally visualises the results.

[9] From [1]

[10] http://www.openk.org

2.4.3 Lightweight coordination calculus

We explain the syntax of LCC in this section. Figure 2.2 defines the syntax used in this thesis.

```
IM            := { Clause, .. }
Clause        := Role :: Def
Role          := a( Type, Id )
Def           := Role | Message | Def then Def | Def or Def | null <- C
Message       := M => Role | M => Role <- C | M <= Role | C <- M <= Role
C             := Term | not C | C and C | C or C
Type          := Term
Id            := Constant | Variable
M             := Term
Term          := Constant | Variable | P( Term, .. )
Constant      := lower case character sequence or number
Variable      := upper case character sequence or number

null: an event which does not involve message passing
<-, not, and, or: the normal logical connectives for implication, negation,
    conjunction and disjunction
```

Figure 2.2: The syntax of lightweight coordination calculus

Shared interaction models are enacted by participants, called peers. The peers play roles on interaction models. Interaction models are written in the Lightweight Coordination Calculus (LCC). The LCC is a lightweight and executable choreography language for specifying coordination among multiple participants based on process calculus.

An LCC interaction model is a set of clauses. Each clause is a definition of a role. Message passing is the only means to transfer information between roles. Sending a message may be conditional on satisfying a constraint and receiving a message may imply constraints on the role to accept it. Those message sending and receiving are the most basic operations. More complex operations are obtained using the connectives (`then` and `or`) for sequence and choices respectively.

Variables, Constants, Terms, Ids and Roles

Variables must start with an upper case letter. The scope of a variable is local to a clause. When it is unnecessary to give a specific name to a variable, you can use an underscore, _, as the variable name. Constants must start with a lower case letter. Numbers also are constants. Terms are either constants, variables or have the form of `P( Term, .. )` where P is a non-numerical constant. Ids are unique identifiers for peers which must be non-numerical constants. Roles are terms that describe the types of roles played by peers in given interactions.

Message

There are two types of messages clauses. Outgoing message clauses have the form of `M => Role`. Incoming message clauses have the form of `M <= Role`. M is the content of the message. The implication operator dominates the message operator: `M => Role <- C` is scoped as `( M => Role ) <- C` and `C <- M <= Role` is scoped as `C <- ( M <= Role )`. Constraint (C) can be attached to both incoming and outgoing forms of message clauses. More details of C are described in the constraints section below.

Constraints

Constraints associate message passing events with conditions established by the peer. Constraints also may be associated with the special null event which represents an event that is not associated with a specific message. This is frequently used in recursive role definition where the role termination depends on a parameter to the role, rather than a specific message passing event.

List operations

List operations are a common basis of the recursion techniques available in the LCC. The bar (|) operation delineates the first element of a list (H, the head) from the rest of the list (T, the tail). That is `L = [ H | T ]`. In the case that H has some value, the constraint of `<- L = [ H | T ]` appends the value of H to the first slot of the list L. In the case that H is not set, the constraint of `<- L = [ H | T ]` extracts H, the value of the first element, from the list L. If L is empty, no value of H is determined and the constraint will fail. Repeated extraction of H

from L and the condition that H has no value realises recursion. The constraint of `<- L = []` is used to test whether L is empty or not.

Logical operators

Constraints can be connected by the logical operators `and` and `or`. `C1 and C2` succeeds if both constraints succeed. `C1 or C2` succeeds if at least one of the constraints succeeds.

Sequence and choice

Sequence (`then`) and choice (`or`) operations determine the sequence of message clauses in a clause. Sequence is written as `E1 then E2`. This sequence is completed when both E1 and E2 are completed. Choice is written as `E1 or E2`. This sequence is completed when either E1 or E2 is completed.

Comments

The double slash comment, `//`, will make the interpreter ignore the rest of the line. The slash-star comment will ignore everything until it meets the next start-slash.

2.5 Social reputation

2.5.1 Network effects

There are situations in which opinions or behaviours of people are affected by other people's opinions or behaviours. This can be for one of two reasons. First, the opinion or behaviours are dependent on others'. Second, other people may give useful opinions or behaviours for decision making. This can derive a different decision from decisions made by independent individuals.

This network effect forms popularity or reputation among individuals. The degree of reputation among them can be extremely imbalanced. The Web is the most appropriate example domain showing the network effect. Reputation of a Web page can be measured with the number of in-links to the Web page.

2.5.2 Power laws

When we measure the distribution of in-links on the Web, we notice that the distribution is different from a normal or Gaussian distribution. Web pages that have k in-links is approximately proportional to $1/k^2$ [14]. They follows a power-law distribution. If the distribution of the Web follows the normal distribution, the number of Web pages with k in-links should decrease exponentially as k grows large. In the power-law distribution, $1/k^2$ decreases much more slowly as k increases. So we can expect that Web pages having a larger number of in-links commonly exist on the power-law distribution than on the normal distribution. This power-law distribution gives a quantitative form to explain why reputation among Web pages is extremely imbalanced.

2.5.3 Rich-get-richer dynamics

Normal distributions may arise from many independent random decisions of individuals. A power-law distribution often arises from feedback by correlated decisions across individuals: a network effect. Here we look into what occurs at the individual decision-making level for a network effect.

Let us go back to the example of Web pages. The following[11] is a simple model of creating links among Web pages.

1. Web pages are created in order and named 1, 2, .., N.
2. When page j is created, the page hangs a link to an earlier Web page according to the following probabilistic rule. Probability p is between 0 and 1.
   (a) With probability p, page j chooses a page i uniformly at random among all pages except itself and hangs a link to the page i.
   (b) With probability $1-p$, page j chooses a page i uniformly at random among all pages except itself and hangs a link to the page to which page i hangs.

   Page j can hang multiple links to other pages through repeating the process of hanging a link.

(b) is the key part. In (b), page j follows page i's opinion about which pages the page j will link to. This opinion copying makes a rich page (having in-links comparatively) get richer (to have more in-links). The degree of getting richer is proportional to the current number of

[11] From [28]

in-links. After it runs for many pages, the fraction of pages with k in-links will be distributed approximately according to a power law $1/k^c$, where the value of c depends on the choice of p.

2.5.4 Pareto principle and the long tail for reputation

The Pareto principle (also known as the 80-20 rule) [67] describes that roughly 80% of the effects come from 20% of the causes. The Pareto principle is one of the instances of a power law effect.

The long tail [2] states that a larger population rests with the tail of a probability distribution than observed under a normal distribution. The long tail is observed in power-law distributions.

Chapter 3

Architectural migration

3.1 Introduction

This chapter provides a smooth architectural migration from traditional ensemble learning systems to an open and social ensemble learning architecture.

3.2 Classifier to classification service

We begin the discussion of migration by explaining how individual classifiers in ensemble systems migrate to classification services in the J-model architecture.

3.2.1 Classifier

3.2.1.1 Structure

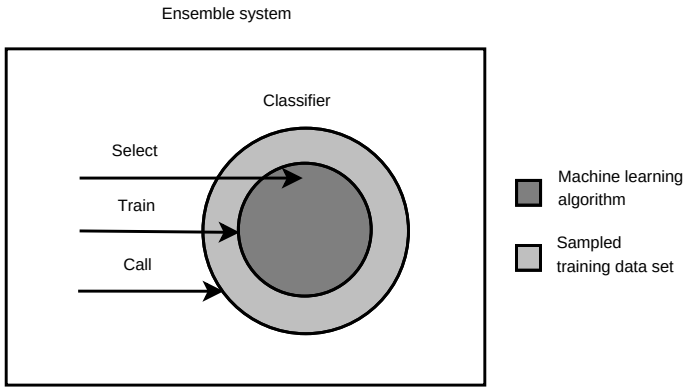

Figure 3.1: A classifier in a traditional ensemble system

To generate a base classifier, an ensemble system selects a model. A model is one of the machine learning algorithms. The selected model is used to do the training with a sampled training data set. A sampled data set may be different for each selected model. The ensemble system repeatedly generates a number of base classifiers, which become the members of an ensemble classifier.

$$c =< m, T > \tag{3.1}$$

A classifier in traditional ensemble learning can be represented as the tuple given in expression (3.1). c is a classifier. m is a model and T is a sampled training data.

3.2.1.2 Properties

In traditional ensemble learning, the members of the ensemble have the following properties:

- Homogeneous $m_1 = m_2$ where $c_1 \neq c_2$

  Classifiers share the same model as their default machine learning algorithm. Diversity and accuracy among classifiers has a trade-off relationship in ensemble learning. Pursuing accuracy is normally better for performance in traditional ensemble systems because pursuing diversity may drive an ensemble to poor classification performance.

- Static $c \rightarrow c$ over time

  A classifier does not change over time. It maintains the same classification performance during an ensemble process. New classifiers are not generated nor are existing classifiers removed during an ensemble process after an initial generation of classifiers.

- Passive

  A classifier does not have ability to communicate with the ensemble system or other member classifiers. A classifier only reacts to requests of prediction from the centrailsed ensemble system (*Call* in Figure 3.1). Its generation process also is completely controlled by the ensemble system (*Select* and *Train* in Figure 3.1).

- Local

  A typical traditional ensemble system is not designed for a distributed environment that classifiers are located in different physical places.

3.2.2 Classification service

3.2.2.1 Structure

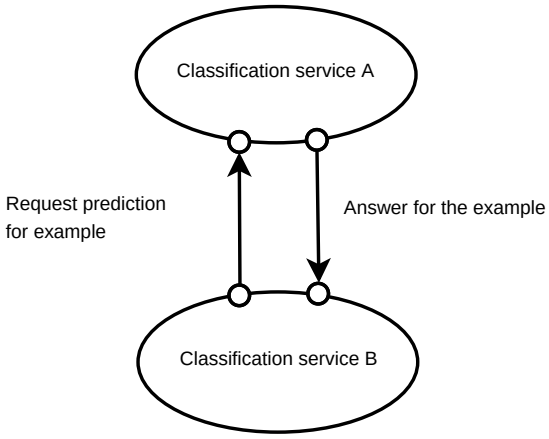

Figure 3.2: A classification service and another classification service

In our open and social ensemble learning approach, classifiers are already present before we apply any coordination method to them. A classifier makes itself available in the form of a service. It has two interfaces: for accepting messages from and sending messages to other classification services. The messages are to request predictions and to give answers for requested predictions as shown in Figure 3.2. Classification services have ability to classify (or are

already trained) regardless of how they got. We consider machine learning classifiers, small reactive devices or even humans as potential classification services.

$$c =< m, T, i, o > \tag{3.2}$$

A classification service can be represented as a tuple equation of (3.2). c is a classification service. m is a model; T: a training data; i: an input interface; o: an output interface.

3.2.2.2 Properties

A classification service has completely nothing to do with a controlling ensemble system. So it has the following properties that are different from the properties of a traditional classifier.

- Heterogeneous $m_1 = m_2$ or $m_1 \neq m_2$ where $c_1 \neq c_2$

  An individual classifier might have a different model from others. The classification services available in a given environment are not chosen in advance to suit the ensemble system.

- Dynamic $c \rightarrow c'$ over time

  A classification service can change over time. There are cases of joining, dismissing, pausing and updating of classification services during an ensemble process.

- Active

  As we mentioned above, a classification service trains data for itself and actively communicates with other classification services using messages.

- Distributed

  A classification service operates on a distributed network or can be on a local machine. Message passing between services can be either over the network or within a local machine.

3.3 Classifier aggregation to service coordination

3.3.1 Classifier aggregation

3.3.1.1 Features on aggregation

3.3.1.1.1 Aggregating all the base classifiers A traditional ensemble method uses all its generated classifiers as members of an ensemble classifier. The members are generated under the supervision of the ensemble method. So it is possible to bring the best performance when using all the classifiers as member classifiers.

$$C = \{c_1, c_2, .., c_N\} \tag{3.3}$$

C is a set of N generated classifiers.

$$M = C \tag{3.4}$$

In a traditional ensemble learning, a set of member classifiers M is identical to C.

3.3.1.1.2 Simple and flat aggregation

$$v(C) = \max_{i=1,2,..,L} \sum_{n=1}^{N} c_{n,i} \tag{3.5}$$

Equation (3.5) represents majority voting that a traditional ensemble normally uses to aggregate predictions from member classifiers.

$c_{n,i}$ is 1 or 0 depending on whether the classifier c_n chooses the class label i or not respectively. The ensemble then chooses a class that receives the largest total vote in L classes.

3.3.1.2 Centralised paradigm

A centralised system means that there is only a single operation process for a task. There is no interaction with other systems. The system freely uses its peripheral units. The units are easily maintained.

A traditional ensemble system is a centralised system. It generates member classifiers. The member classifiers do their roles as sub functions in the system. The system gathers and aggregates predictions from the member classifiers. Then the system gives a final prediction answer.

3.3.1.3 UML activity diagram

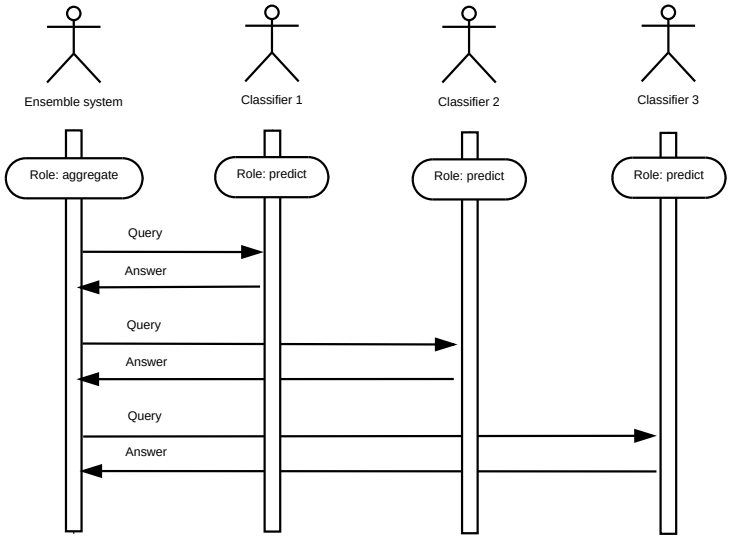

Figure 3.3: UML activity diagram of traditional ensemble aggregation

Figure 3.3 is an example of a UML activity diagram that describes a typical traditional ensemble aggregation process. The ensemble system actor takes the *aggregate* role. All of the classifier actors take the *predict* role. The ensemble system actor and the classifier actors run from the beginning to the end of an aggregation process. The ensemble system actor sends query messages to the classifier actors and gets answer messages from them serially. The ensemble system actor controls this process centrally.

3.3.2 Service coordination

Service coordination is a means to weave predictions from classification services in our open and social ensemble learning. Protocols coordinate the actions of classification services.

3.3.2.1 Service coordination and its elements

Service coordinations impose constraints on the interactions between services for particular applications. When services are being coordinated, a coordination context propagates to the services and by committing to participate in the interaction described by the protocol they also accept the constraints it imposes on their roles in the protocol. Coordination context flows by message exchange among services. We introduce the elements of service coordination and what each element means in our classification service coordination.

- Participant
  When a service is enacted in a coordination, a service is called a participant. In classification service coordination, classification services and coordinators are participants.

- Role
  A classification service can take a role of predicting query and test examples. A coordinator boots a coordination process and gives a final prediction answer from predictions of classification services.

- Activity
  Participants communicate with one another through message exchange. Message sending and accepting are activities.

- Message
  Messages in classification service coordination are requests for predictions and prediction answers.

- Coordinator
  A coordinator is needed to ensure that the protocols selected by participants are made available to the relevant participants and that the roles accepted by each participant are discharged according to the protocol. Coordinators need not be centralised - they may be distributed across the services or, in the most extreme case, distributed with the messages passed between services.

- Protocol
  A protocol defines roles and activities. A traditional ensemble system conceptually can be mapped to a protocol. We script protocols with lightweight coordination language (LCC), a service choreography language explained in Section 2.4.3 for our classification service coordination. We call a protocol as an *interaction model* in our classification service coordination. We show multiple actual interaction models for classification in

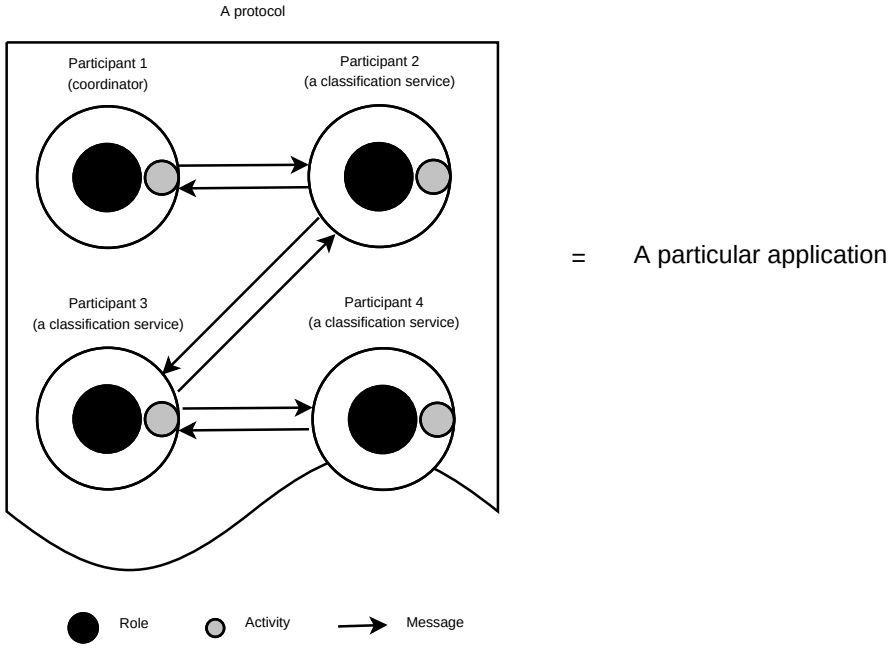

Figure 3.4: Classification service coordination and its elements

Figure 3.4 presents a visual organisation of the service coordination elements.

3.3.2.2 Choreography paradigm

Our classification service coordination is based on service choreography paradigm. No participant controls the protocol centrally. Once an interaction model is published to participants, participants take their roles individually and exchange messages one another as defined in the interaction model. When message exchange is completed, the service playing the role of coordinator gives the final prediction.

3.3.2.3 UML activity diagram

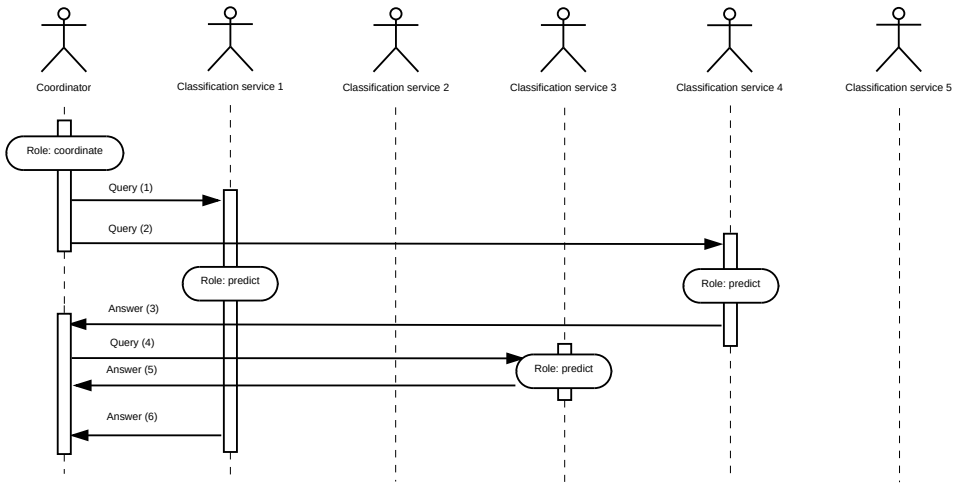

Figure 3.5: UML activity diagram of classification service coordination

Figure 3.5 is a simple example of a UML activity diagram that represents classification service coordination. The coordinator actor takes the *coordinate* role and classification service actors take the *predict* role. Actors only run while they send or receive messages. Message exchange is asynchronous. In the example, some classification services (classification services 2 and 5) do not take any role. This might be because they cannot satisfy the required roles or that all of the roles of the interaction model have already be taken.

3.3.2.4 Representation and features

Our open and social ensemble learning method considers all existing classification services as potential participants for coordination.

$$C = \{c_1, c_2, .., c_N\} \tag{3.6}$$

C is a set of N existing classification services.

$$M \subseteq C \tag{3.7}$$

Participating classification services M is the same as C or is a subset of C. This is determined according to how many classification services satisfy roles defined in a coordination protocol or how many roles a coordination protocol defines.

$$E =< IM, M > \tag{3.8}$$

An ensemble classification service can be represented as the tuple of expression (3.8). E is an ensemble classification service and IM is an interaction model. An ensemble is an interaction model supported by participating classification services M.

We can freely define interaction models to coordinate learners in an ensemble using the LCC service choreography language. This allows complex and perhaps domain specific interactions to be implemented as well as the standard protocols for traditional ensemble learning. It is then the responsibility of the service enactment system to ensure, as reliably as is reasonably possible, that the right learners adopt appropriate roles in the coordinated ensemble - leaving engineers with the task of defining coordination rather than defining specific ensembles each time.

3.4 Performance boosting strategies

There are two main approaches to boost performance of an ensemble classifier in ensemble learning. One is to prepare better base classifiers. The other is to coordinate base classifiers better.

3.4.1 In traditional ensemble learning

In traditional ensemble learning, boosting performance is achieved by preparing better base classifiers through generating more diverse classifiers and more accurate classifiers. Krogh and Vedelsby [53] have formally shown why a better ensemble classifier can be obtained from more diverse and more accurate base classifiers. As we showed in the previous section 3.3, a traditional ensemble method does not support complex coordinations for classifiers.

3.4.1.1 More diverse classifiers

Bagging is a representative ensemble method to build an ensemble classifier through generating diverse base classifiers. We show the Bagging algorithm from Section 2.2 here again to explain how it generate diverse base classifiers.

Algorithm 3.1 More diverse classifiers in Bagging

Require: I (a base inducer), T (number of iterations), S (the original training set), μ (the sample size)

1: $t \leftarrow 1$

2: **repeat**

3: $S_t \leftarrow$ a sample of μ instances from S with replacement.

4: Construct classifier M_t using I with S_t as the training set

5: $t \leftarrow t + 1$

6: **until** $t > T$

In line 3, a sampled data set S_t is made from the original set S with replacement. Then a classifier M_t trains the sampled data set S_t. From those steps, diverse base classifiers can be generated.

3.4.1.2 More accurate classifiers

Algorithm 3.2 More accurate classifiers in Boosting

Require: I (a weak inducer), S (training set) and k (the sample size for the first classifier)

Ensure: M_1, M_2, M_3

1: $S_1 \leftarrow$ Randomly selected $k < m$ instances from S without replacement;

2: $M_1 \leftarrow I(S_1)$

3: $S_2 \leftarrow$ Randomly selected instances (without replacement) from $S - S_1$ such that half of them are correctly classified by M_1.

4: $M_2 \leftarrow I(S_2)$

5: $S_3 \leftarrow$ any instances in $S - S_1 - S_2$ that are classified differently by M_1 and M_2.

6: $M_3 \leftarrow I(S_3)$

Boosting is an example of a method for making more accurate base classifiers. We also show the Boosting algorithm of Section 2.2 here again to describe how it generates accurate base classifiers.

In line 1, a sampled set S_1 is made through a random instance selection. In line 2, a temporal classifier M_1 is generated from S_1. In the 3rd and 4th lines, S_2 is a weighted set for misclassified instances and another temporal classifier M_2 is generated from S_2. In line 5, a more weighted set for misclassified instances S_3 is prepared. From those steps, more accurate base classifiers are generated.

3.4.2 In J-model

3.4.2.1 Recommendation of services

The first approach for boosting classification performance in J-model is to use more appropriate classification services for classification among existing classification services. This is implemented by a recommendation mechanism called the peer ranking algorithm [79, 57] that we describe in Chapter 5 in detail.

$$M_t \subseteq C \tag{3.9}$$

M_t is a set of classification services as members of an ensemble classifier at the current interaction t.

$$E_t =< IM, M_t > \tag{3.10}$$

E_t is an ensemble classifier supported by M_t at the tth interaction. If E_t gives good performance, M_t gets a higher reputation for classification. If E_t gives bad performance, M_t gets a lower reputation for classification. For each subsequent interaction M_t is composed of classifiers chosen to have the highest reputation. As this process iterates, we expect better member classifiers to be identified, thus reinforcing the quality of classification overall.

3.4.2.2 Interaction model design

The other approach for boosting performance is to coordinate member classification services better. This is implemented by programming a better designed interaction model for an individual classification task.

The aggregation of a traditional ensemble learning is the simplest form of coordination. Meanwhile we can design various effective interaction models for different specific classification purposes in J-model such as getting better specificity, achieving higher true positive rate and reducing classification time cost. Of course we can programme more complex interaction models for a general performance measure such as *accuracy* reflecting the features of a classification data set. So interaction models are plural, not single in J-model. We suggest examples of specifically designed interaction models in Chapter 6.

$$IM \in \{IM_1, IM_2, ..\} \tag{3.11}$$

An interaction model to be used for a task (IM) is an instance of a source interaction model from the set $\{IM_1, IM_2, ..\}$.

Chapter 4

J-model architecture

4.1 Introduction

In the previous chapter, we explained the relationship between traditional ensemble methods and the J-model approach. We now describe the J-model architecture in more detail, based on the architectural migration shown in Chapter 3.

4.2 Organisation of J-model

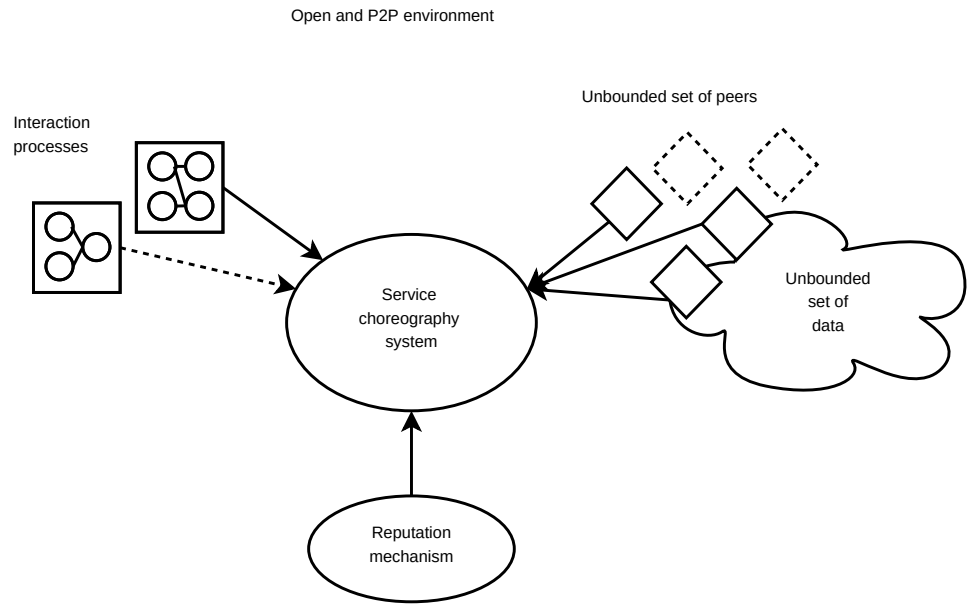

Figure 4.1: J-model architecture

Figure 4.1 illustrates the J-model architecture and its components. J-model is a service choreography system for coordination of services (peers in the figure) who exist in open environments. Interaction processes defines protocols in which peers interact in a peer-to-peer manner. Reputation mechanism recommends adequate peers who will participate in the choreographies. We explain each J-model component in detail in the following sub-sections.

4.2.1 Service choreography system

The service choreography system supplies the infrastructure upon which the other parts of interaction processes, set of peers and data and reputation mechanism work together.

The service choreography system is not executed. It is enacted. A chosen interaction process or *interaction model* for a specific task is published (introduced in Section 3.3.2). Peers or *classification services* take roles and exchange message with a peer to peer approach on the interaction model (explained in Section 3.2.2). A reputation mechanism such as *the peer ranking algorithm* given in this thesis evaluates the participating peers and then recommends more appropriate peers for the interaction model (introduced in Section 3.4.2).

4.2.2 Interaction processes

What is usually called an interaction process in service-oriented architectures is an *interaction model* in J-model. An interaction model (IM) defines roles for classification services and what messages they will exchange. An interaction model is shared among classification services. In J-model, interaction models are specified in the lightweight coordination calculus (LCC) language. We show definitions of interaction models for ensemble classification in Chapter 6.

4.2.3 Set of peers and data

Peers or *classification services* are services that have the ability to perform classification. They are not engaged with the service choreography system until an interaction model is published among them. When an interaction model is published, each classification service takes a role in the interaction model. Classification services may not always be machine learners. Classifiers may include small and reactive devices or even humans.

Data that classification services train are also distinct from their classifiers. Data may be changing over time. Also different classification services can become engaged with the service choreography system over time.

4.2.4 Reputation mechanism

A reputation mechanism is an algorithm to recommend more appropriate peers to the service choreography system. An interaction model can achieve its task better with the recommended peers. In J-model, we use as a reputation mechanism *the peer ranking algorithm* defined in Chapter 5.

4.3 Discovery-enactment-analysis cycle

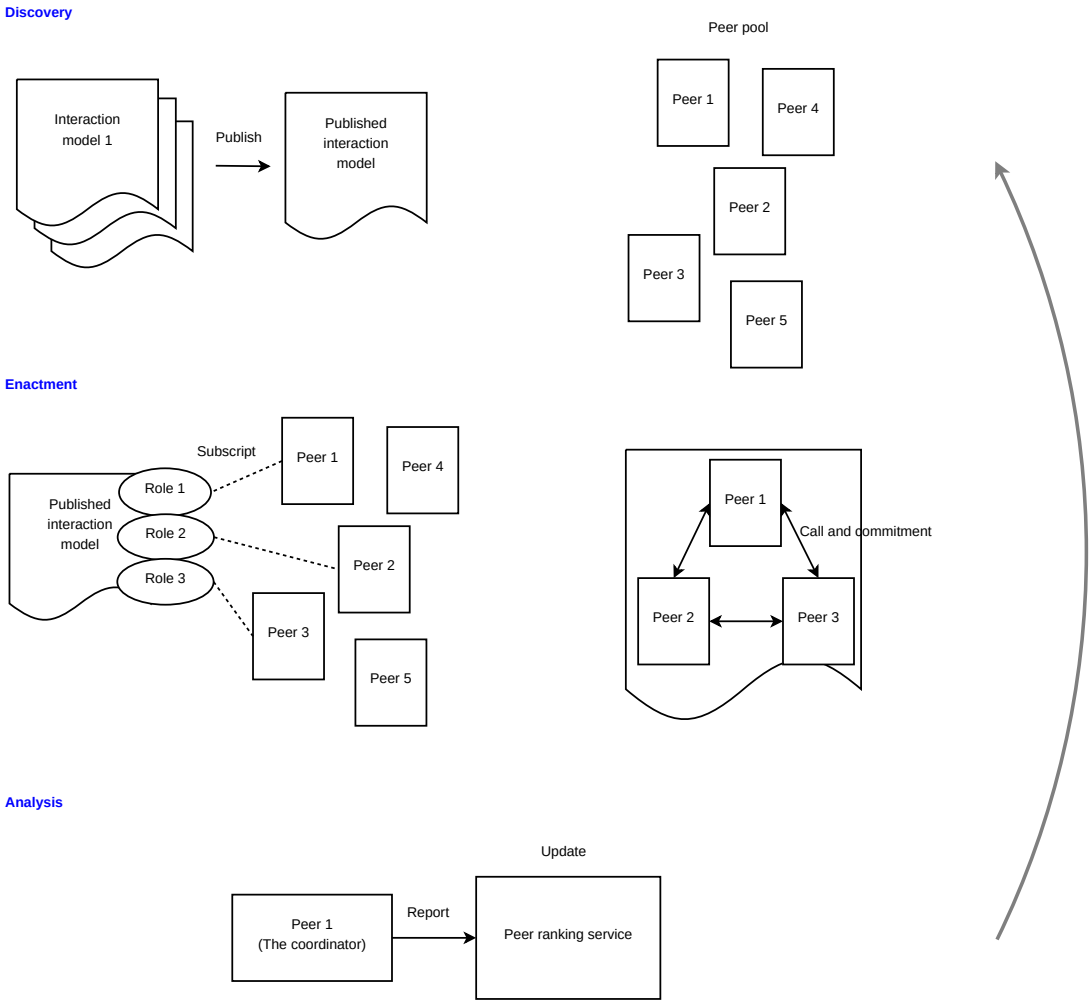

Figure 4.2: Discovery-enactment-analysis cycle in J-model architecture

We analyse the J-model architecture from an operational viewpoint (so we use *phase* as a term for individual processes in J-model). This can be divided three phases; discovery, enactment and analysis. These operate cyclically because the peer ranking service in the analysis phase updates the rank information for the peers.

We describes this discovery-enactment-analysis cycle with a pseudocode as follows.

Require: *ims* (a set of interaction models), *peers* (a pool of peers), *ensemble_size* (the size of ensemble classifier) and *n* (the number of interactions)
// Discovery phase
Select *im* **In** *ims*
Publish *im* **To** *peers*
// Enactment phase
Select Randomly *participants* **Size Of** *ensemble_size* **In** *peers*
n **Times Do**
    For Each *role* **In** (*roles* **Defined In** *im*) **Do**
        Select *participant* **For** *role* **In** *participants*
        participant **Subscript** *role*
    End
    result ← **Commit** *participants*
    // Analysis phase
    Report *result* **To** *peer_ranking_service*
    peer_ranking_service **Recommend** *participants*
End

4.4 Training, query and test layers

We can analyse J-model based on a typical machine learning process that includes training and testing steps. We use *layer* for the individual steps instead of *phase* as we illustrate J-model from an architectural viewpoint.

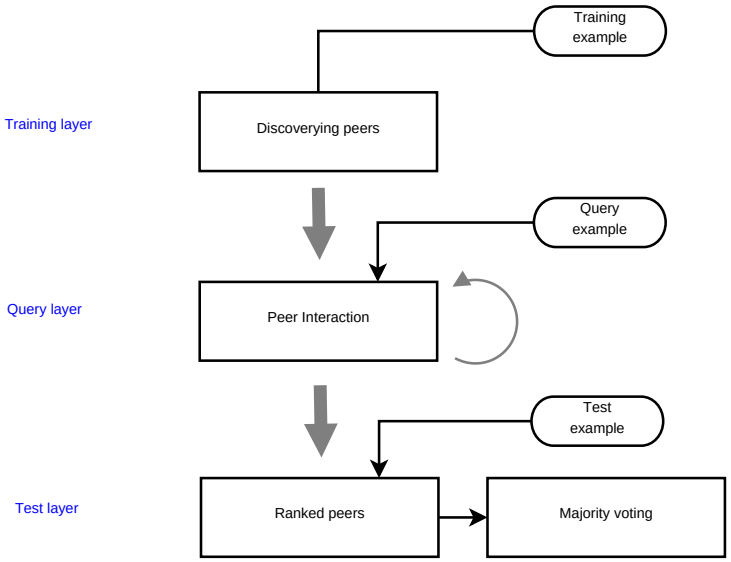

Figure 4.3: Training, query and test layers in J-model architecture

4.4.1 Training layer

In a typical machine learning process, the training step prepares training examples and build a model (hypothesis) that trains the training examples. In traditional ensemble learning, two or more training sets and models are generated.

In J-model, peers are already trained classification services regardless of how and on what they were trained. To discover peers is the training layer from a traditional viewpoint.

4.4.2 Query layer

The query layer is a unique part that is found only in J-model. This layer is mapped to the enactment and analysis phases in Figure 4.2.

In a typical machine learning process, an original data set is split into training and test data sets. J-model additionally needs a query data set. A query example is used to evaluate participating peers in a current interaction model. If the evaluation is a success (the interaction model achieved its goal. e.g. giving a correct prediction for the query example), reputation of the peers rises. We show later how the reputation distribution for peers converges through repeated evaluation with query examples.

4.4.3 Test layer

In the test layer, top-ranked peers of the ensemble size that we set become members of an ensemble classifier. They vote to predict answers for test examples using majority voting.

Chapter 5

Peer ranking service

5.1 Introduction

A peer ranking service recommends those classification services for an interaction model that have higher success rate. This allows J-model to improve its classification performance.

5.2 Recommendation-evaluation-update cycle

We now describe peer recommendation in J-model focusing on the role of the peer ranking service and the messages from and to the service.

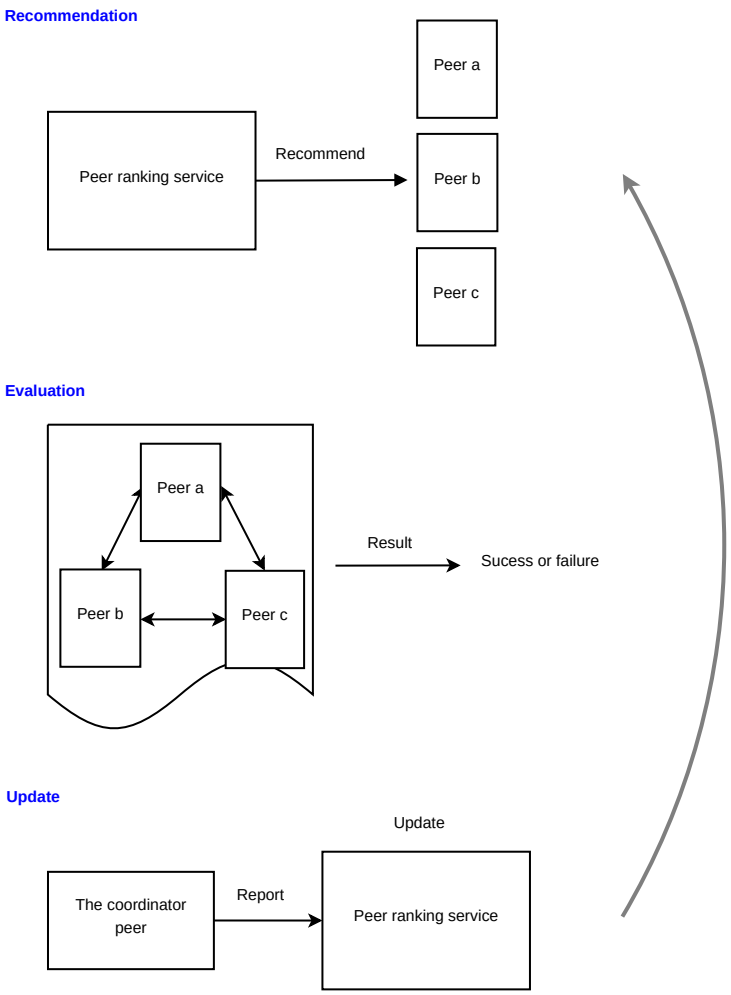

Figure 5.1: Recommendation-evaluation-update cycle on peer ranking service

5.2.1 Recommendation phase

The peer ranking service recommends top-ranked peers based on the rank distribution of peers that the service maintains on their roles in interaction models with which they have previously engaged. In Figure 5.1, Peers *a*, *b* and *c* in a peer pool was recommended by the peer ranking service.

5.2.2 Evaluation phase

Recommended peers interact with one another on a interaction model for a query example. The result is a success or failure. A success is that the interaction achieves its defined goal. We show examples of interaction models for classification including their various goals in Chapter 6.

5.2.3 Update phase

The coordinator peer reports the interaction result to the peer ranking service. The peer ranking service updates the ranking scores of the current recommended peers. The rank distribution of peers is then re-calculated for the next recommendation phase.

5.3 Peer ranking algorithm

The peer ranking algorithm recommends peers that will take roles in an interaction model. For query interactions, the peer ranking algorithm needs to recommend peers more exploratively to avoid becoming stuck in a local optimum. For a test interaction, the algorithm needs to recommend peers reflecting all the history of scores that peers have individually. This guides that the peer ranking algorithm can suggest a global optimum or at least a better local optimum for tests.

Algorithm 5.1 The peer ranking algorithm for query interactions

Require: P (a peer pool), T (the number of interactions), IM (interaction model), N (the ensemble size), Q (a set of query examples)

01: $C(p,\ominus) \leftarrow 0$ for each peer p in P

($C(p,\ominus)$ is the count of minus that p has.)

02: $t \leftarrow 1$

03: **repeat**

04: $M_t \leftarrow$ Pick the highest ranked N peers in P based on the query rank calculation R_Q

($R_Q(p) = C(p,\ominus)$. A lower value of $R_Q(p)$ means that p obtains a higher rank.)

05: $E_t \leftarrow < IM, M_t >$

06: $A_t \leftarrow$ Evaluate E_t with a randomly chosen query example from Q

07: **if** $A_t = success$

08: Increase the count of plus that each m_i has where $m_i \in M_t$

09: **else if** $A_t = failure$

10: Increase the count of minus that each m_i has where $m_i \in M_t$

11: **end if**

12: $t \leftarrow t+1$

13: **until** $t > T$

$A \in \{success, failure\}$

Algorithm 5.1 is for recommending peers for query interactions. The algorithm evaluates an ensemble composed of recommended peers with a query example at each interaction round.

Algorithm 5.2 The peer ranking algorithm for test interaction

Require: P (a peer pool), IM (interaction model), N (the ensemble size), s (a test example)

01: $M \leftarrow$ Pick the highest ranked N peers in P based on the test rank calculation R_T

($R_T(p) = C(p,\ominus)/C(p,\oplus)$. $C(p,\oplus)$ is the count of plus that p has. A lower value of $R_T(p)$ means that p obtains a higher rank.)

02: $E \leftarrow < IM, M >$

03: $A \leftarrow$ Evaluate E with s

$A \in \{success, failure\}$

Algorithm 5.2 is for recommending peers for a test interaction. An ensemble composed of recommended peers is used to predict test examples.

5.4 Examples

We show how the peer ranking algorithm works with two examples. The examples follows changes of the rank distribution of peers. One example is under static conditions. The other is under dynamic conditions. Under static conditions, all the peers permanently exist and their learning status does not change while the peer ranking algorithm applies to the peers. In contrast to static conditions, peers can attend, dismiss and be changed for their learning status under dynamic conditions.

5.4.1 Under static conditions

Table 5.1: Peer ranking example under static conditions

# of interactions	Recommended peers	Result of completion	p_0	p_1	p_2
Initial status	-	-	<0, 0> (1)	<0, 0> (1)	<0, 0> (1)
1	p_0, p_1	Success	<1, 0> (1)	<1, 0> (1)	<0, 0> (1)
2	p_1, p_2	Failure	<1, 0> (1)	<1, 1> (2)	<1, 1> (2)
3	p_0, p_2	Success	<2, 0> (1)	<1, 1> (2)	<2, 1> (2)
4	p_0, p_1	Success	<3, 0> (1)	<2, 1> (2)	<2, 1> (2)
5	p_0, p_2	Failure	<3, 1> (1)	<2, 1> (1)	<2, 2> (2)
6	p_0, p_1	Success	<4, 1> (1)	<3, 1> (1)	<2, 2> (2)
7	p_0, p_1	Failure	<4, 2> (1)	<3, 2> (1)	<2, 2> (1)
8	p_0, p_1	Failure	<4, 3> (1)	<3, 3> (2)	<2, 2> (1)
9	p_2, p_1	Success	<4, 3> (2)	<4, 3> (2)	<3, 2> (1)
10	p_2, p_0	Success	<5, 3> (2)	<4, 3> (2)	<4, 2> (1)
For test	p_2, p_0	-	$3/5 = 0.6$ (2)	$3/4 = 0.75$ (3)	$2/4 = 0.5$ (1)

In this example, 10 interactions are applied for three peers of p_0, p_1 and p_2. The first element of a tuple is the number of pluses that a specific peer has. The second element of a tuple is the number of minuses that a specific peer has. The rank of a specific peer is in parentheses.

Recommended peers on each interaction are selected based on their current rankings. For example, after the 6th interaction finishes, p_0 and p_1 are recommended for the next interaction because they are the higher ranked peers.

After all the interactions finish, p_2 and p_0 are recommended for test.

5.4.2 Under dynamic conditions

Table 5.2: Peer ranking example under dynamic conditions

# of interactions	Recommended peers	Result of completion	p_0	p_1	p_2
Initial status	-	-	<0, 0> (1)	<0, 0> (1)	<0, 0> (1)
1	p_0, p_1	Success	<1, 0> (1)	~~<1, 0> (1)~~	<0, 0> (1)
2	p_0, p_2	Success	~~<2, 0> (1)~~	<1, 0> (1)	<1, 0> (1)
3	p_1, p_2	Failure	<2, 0> (1)	<1, 1> (2)	~~<1, 1> (2)~~
4	p_0, p_1	Success	<3, 0> (1)	<2, 1> (2)	~~<1, 1> (2)~~
5	p_0, p_1	Failure	~~<3, 1> (1)~~	<2, 2> (2)	<1, 1> (1)
6	p_2, p_1	Success	<3, 1> (1)	<3, 2> (2)	~~<2, 1> (1)~~
7	p_0, p_1	Success	<4, 1> (1)	<4, 2> (2)	~~<2, 1> (1)~~
8	p_0, p_1	Failure	~~<4, 2> (2)~~	<4, 3> (3)	<2, 1> (1)
9	p_2, p_1	Success	<4, 2> (2)	<5, 3> (3)	~~<3, 1> (1)~~
10	p_0, p_1	Failure	<4, 3> (2)	<5, 4> (3)	<3, 1> (1)
For test	p_2, p_0	-	$3/4 = 0.75$ (2)	$4/5 = 0.8$ (3)	$1/3 = 0.33$ (1)

In this example, 10 interactions are also applied for three peers of p_0, p_1 and p_2. We intentionally made one the three peers be unavailable randomly at each interaction (represented as strikethrough on a peer).

For the 6th interaction, p_2 and p_1 are recommended although p_0 has a higher ranking than p_1 because p_0 is unavailable. Absence of a peer at each interaction effects the rank distribution of peers. Different peers from the static example (Table 5.1) finally can be members of an ensemble.

5.5 Termination condition

Typically we can set when an interaction needs to finish or how many interactions are needed to get a good ensemble with the following two methods.

5.5.1 Number of interactions

To set the number of interactions is a static approach. Computation is terminated after the fixed number of interactions. We can stop the convergence at the point we want. J-model, however, might not be sufficiently converged at that point.

5.5.2 Performance metric criterion

We can set an expected performance value on a performance metric. For example, if we expect that J-model should give 80% accuracy, we wait until J-model's prediction converges to 80% accuracy over an appropriate number of interactions. When an ensemble at each interaction reaches the expected value, interaction finishes. We may get the expected performance with this method, but if the expected value is too strict then J-model might never converge.

Chapter 6

Interaction models for classification

6.1 Introduction

This chapter describes how lightweight coordination calculus (LCC) scripts interaction models (IMs) for an open environment and shows examples for different forms of classification.

Definition 6.1. *Closed simple interaction model*

```
a( ensemble_classifier( TestExample, Peers, Result ), E ) ::
    a( coordinator( TestExample, Peers, Answers ), R ) then
    null <- vote( Answers, Result )

a( coordinator( TestExample, Peers, Answers ), R ) ::
    (
    ask( TestExample ) => a( classifier, C ) <- Peers = [ C | RestPeers ] then
    answer( Answer ) <= a( classifier, C ) then
    a( coordinator( TestExample, RestPeers, Answers ), R ) <- Answers = [ Answer | RestAnswers ]
    )
    or
    null <- Peers = []

a( classifier, C ) ::
    ask( TestExample ) <= a( coordinator( TestExample, Peers, Answers ), R ) then
    answer( Answer ) => a( coordinator( TestExample, Peers, Answers ), R ) <-
        predict( TestExample, Answer )
```

Definition 6.1 is a very simple form of interaction model for ensemble classification. The IM has three roles: `ensemble_classifier` (`E`), `coordinator` (`R`) and `classifier` (`C`). The ensemble classifier gets prediction answers (`Answers`) for a test example (`TestExample`) from the coordinator and calculates the voted answer (`Result`) from the prediction answers. The coordinator recursively asks classifiers (`Peers`) to get predicted answers for the test example

from them. A classifier takes a message that is a request to predict an answer from the coordinator. The `predict` constraint predicts an answer on the peer acting as classifier and the classifier sends back the answer message to the coordinator. This interaction model is written using LCC, considers the classifiers as services and runs these as decentralised processes. This interaction model, however, is actually the same as a traditional ensemble method as far as its behaviour is concerned. Participating classifiers are fixed and all the predicted answers from them are just simply aggregated for the ensemble.

Definition 6.2. *Closed complex interaction model*

```
a( ensemble_classifier( TestExample, Peers, S, T, Result ), E ) ::
    a( meta_coordinator( TestExample, S, T, Peers, Answers ), M ) then
    null <- vote( Answers, Result )

a( meta_coordinator( TestExample, S, T, Peers, Answers ), M ) then
    (
    a( coordinator( TestExample, PPeers, PAnswers ), R ) <-
        ( S > 0 and pick_peers( T, Peers, PPeers, RestPeers ) ) then
    a( meta_coordinator( TestExample, S1, T, RestPeers, Answers ), M ) <-
        S1 is S - 1 and vote( PAnswers, Result ) and Answers = [ Result | RestAnswers ]
    )
    or
    null <- S = 0

a( coordinator( TestExample, Peers, Answers ), R ) ::
    (
    ask( TestExample ) => a( classifier, C ) <- Peers = [ C | RestPeers ] then
    answer( Answer ) <= a( classifier, C ) then
    a( coordinator( TestExample, RestPeers, Answers ), R ) <- Answers = [ Answer | RestAnswers ]
    )
    or
    null <- Peers = []

a( classifier, C ) ::
    ask( TestExample ) <= a( coordinator(TestExample, Peers, Answers ), R ) then
    answer( Answer ) => a( coordinator( TestExample, Peers, Answers ), R ) <-
        predict( TestExample, Answer )
```

We can design a more complex coordination interaction to the closed simple IM of Definition 6.1. LCC can programme complex interaction models that are tuned at specific coordination strategies or for particular problem domains. Definition 6.2 shows an example of a more complex interaction model.

The IM defines four roles: `E`, `meta_coordinator` (`M`), `R` and `C`. The ensemble classifier role here is a slightly modified version of E in the closed simple IM example. The meta coordinator which is now introduced is for more complex coordination of classifiers. `R` and `C` are the same

ones as in the closed simple IM example.

The ensemble classifier passes the sequence to the meta coordinator (`meta_coordinator( TestExample, S, T, Peers, Answers ), M )`) instead of passing it to the coordinator. `S` is a parameter to set the number of voters. Each voter aggregates classifier's predictions independently. `T` is the number of classifiers that give back prediction answers to a voter. The `vote` constraint of the ensemble classifier aggregates the voted answers of the voters of `M`.

The meta coordinator picks classifiers (`pick_peers( T, Peers, PPeers, RestPeers )`) and passes the sequence to the coordinator. `Answers` in the meta coordinator is the collection of voted answers from the coordinator with the picked classifiers.

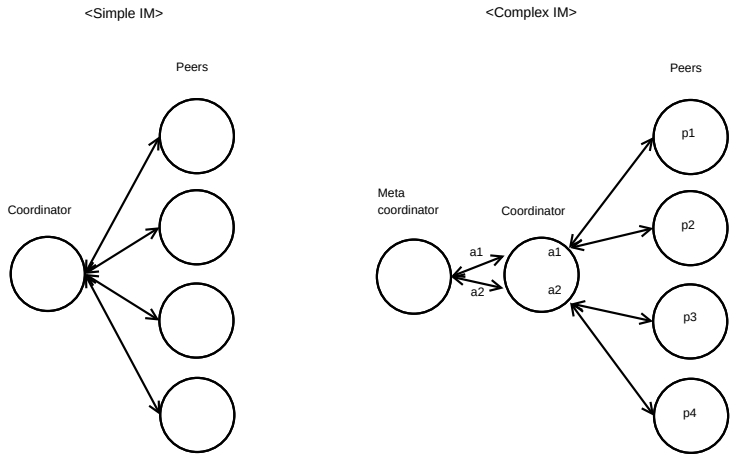

Figure 6.1: Simple and complex interaction models

Figure 6.1 describes how the simple IM and the complex IM coordinate classifiers. In the simple IM, the coordinator gets answers from all the peers and votes the answers. In the complex IM, the coordinator gets answers from `T` picked peers and votes the answers `S` times. The meta coordinator gets `S` voted answers. In the figure, `T` is 2 and `S` is 2. $[p_1, p_2]$ is a set of picked peers and $[p_3, p_4]$ is the other set of picked peers. One voted answer (a_1) is from $[p_1, p_2]$ and the other (a_2) is from $[p_3, p_4]$.

Here we need to pay attention to those two IMs. They fix their participating classifiers (`Peers`) when they are deployed at design time. This means that which classifiers will participate for the ensemble has already been pre-determined. These IMs are for a closed classifier environment. Instead we need other forms of interaction model that can coordinate classifiers in an open environment. In the following section, we suggest interaction models for an open environments.

6.2 Open interaction models

6.2.1 Simple model (IM1)

In the open classifier environment, participating classifiers are not fixed. So we cannot determine which classifiers participate for ensemble when an IM is deployed. We can only set the number of classifiers which will be the members of an ensemble classifier. Actual member classifiers are determined as they interact in the IM at run-time.

Definition 6.3. *ensemble_classifier role and choose_peers constraint*

```
a( ensemble_classifier( TestExample, N, Result, Peers ), E ) ::
    a( coordinator( TestExample, Peers, Answers ), R ) <- choose_peers( N, Peers ) then
    null <- vote( Answers, Result )
```

The `ensemble_classifier` role in Definition 6.3 takes only the number of classifiers (`N`) as its parameter instead of the member classifiers themselves (`Peers` in the closed interaction models). Actual peers are determined for their roles by the `choose_peers` constraint. `choose_peers` recommends N peers to the coordinator.

Definition 6.4. *Open simple interaction model without peer ranking*

```
a( ensemble_classifier( TestExample, N, Result, Peers ), E ) ::
    a( coordinator( TestExample, Peers, Answers ), R ) <- choose_peers( N, Peers ) then
    null <- vote( Answers, Result )

a( coordinator( TestExample, Peers, Answers ), R ) ::
    (
    ask( TestExample ) => a( classifier, C ) <- Peers = [ C | RestPeers ] then
    answer( Answer ) <= a( classifier, C ) then
    a( coordinator( TestExample, RestPeers, Answers ), R ) <- Answers = [ Answer | RestAnswers ]
    )
    or
    null <- Peers = []

a( classifier, C ) ::
    ask( TestExample ) <= a( coordinator( TestExample, Peers, Answers ), R ) then
    answer( Answer ) => a( coordinator( TestExample, Peers, Answers ), R ) <-
        predict( TestExample, Answer )
```

Definition 6.4 is an open interaction model example. It includes `ensemble_classifier` role of Definition 6.3. The coordinator and classifier roles are the same as with the closed interaction models.

Definition 6.5. *peer_ranker role*

```
a( peer_ranker( QueryExample, QClass, N ), K ) ::
    a( ensemble_classifier( QueryExample, N, Result, Peers ), E ) then
    null <- update( Result, QClass, Peers )
```

We have designed an open interaction model that, through the `N` parameter, selects `N` peers using the `choose_peers` constraint. This selection, however, does not guarantee that the chosen peers are appropriate members for the ensemble. The ensemble classifier from the chosen classifiers may not give adequate classification performance because `choose_peers` does not have any criterion upon which peers to choose. Therefore we need to make `choose_peers` get peers based on the recommendation of the peer ranking service. As we have shown in Chapter 5, the peer ranking algorithm iteratively assigns a measure of reputation to members of the ensemble.

The `peer_ranker` role in Definition 6.5 reports the prediction results of currently chosen peers to the peer ranking service. A query example (`QueryExample`) is a query for which each chosen peer answers with a prediction of its class label. `QClass` is the actual or correct class label for the query. `N` is the number of peers to be chosen as members. `peer_ranker` passes its sequence to `ensemble_classifier` with `QueryExample` and gets a voted answer (`Result`) and the chosen peers (`Peers`). `update` constraint compares the predicted answer (`Result`) with the correct class (`QClass`) and reports the comparison results to the peer ranking service. `peer_ranker` is an evaluator that evaluates the prediction from the ensemble classifier. Iterative calling of `peer_ranker` updates the reputation of the classifiers in the open environment. Consequently `choose_peers` can recommend more appropriate peers.

Definition 6.6. *Open simple interaction model with peer ranking (IM1)*

```
a( ensemble_classifier( TestExample, N, Result, Peers ), E ) ::
    a( coordinator( TestExample, Peers, Answers ), R ) <- choose_peers( N, Peers ) then
    null <- vote( Answers, Result )

a( coordinator( TestExample, Peers, Answers ), R ) ::
    (
    ask( TestExample ) => a( classifier, C ) <- Peers = [ C | RestPeers ] then
    answer( Answer ) <= a( classifier, C ) then
    a( coordinator( TestExample, RestPeers, Answers ), R ) <- Answers = [ Answer | RestAnswers ]
    )
    or
    null <- Peers = []

a( classifier, C ) ::
    ask( TestExample ) <= a( coordinator( TestExample, Peers, Answers ), R ) then
    answer( Answer ) => a( coordinator( TestExample, Peers, Answers ), R ) <-
        predict( TestExample, Answer )

a( peer_ranker( QueryExample, QClass, N ), K ) ::
```

```
    a( ensemble_classifier( QueryExample, N, Result, Peers ), E ) then
    null <- update( Result, QClass, Peers )
```

Definition 6.6 shows all the roles that we introduced in this section in one place.

6.2.2 Complex model (IM2)

We do not need to be limited to a simple model. We can design other interaction models that are tuned to diverse coordination strategies or particular domains in the open classifier environment.

Definition 6.7. *Open complex interaction model 1 (IM2)*

```
a( ensemble_classifier( TestExample, N, S, T, Result, Peers ), E ) ::
    a( meta_coordinator( TestExample, S, T, Peers, Answers ), M ) <- choose_peers( N, Peers ) then
    null <- vote( Answers, Result )

a( meta_coordinator( TestExample, S, T, Peers, Answers ), M ) then
    (
    a( coordinator( TestExample, PPeers, PAnswers ), R ) <-
        ( S > 0 and pick_peers( T, Peers, PPeers, RestPeers ) ) then
    a( meta_coordinator( TestExample, S1, T, RestPeers, Answers ), M ) <-
        S1 is S - 1 and vote( PAnswers, Result ) and Answers = [ Result | RestAnswers ]
    )
    or
    null <- S = 0

a( coordinator( TestExample, Peers, Answers ), R ) ::
    (
    ask( TestExample ) => a( classifier, C ) <- Peers = [ C | RestPeers ] then
    answer( Answer ) <= a( classifier, C ) then
    a( coordinator( TestExample, RestPeers, Answers ), R ) <- Answers = [ Answer | RestAnswers ]
    )
    or
    null <- Peers = []

a( classifier, C ) ::
    ask( TestExample ) <= a( coordinator( TestExample, Peers, Answers ), R ) then
    answer( Answer ) => a( coordinator( TestExample, Peers, Answers ), R ) <-
        predict( TestExample, Answer )

a( peer_ranker( QueryExample, QClass, N ), K ) ::
    a( ensemble_classifier( QueryExample, N, Result, Peers ), E ) then
    null <- update( Result, QClass, Peers )
```

Definition 6.7 is an example of open complex IM. It has adopted the `meta_coordinator` role and `S` and `T` parameters from the open simple IM in the same way that the closed complex IM adopted those elements. The details of `meta_coordinator`, `S` and `T` are the same as with the

closed complex IM's.

6.2.3 Another complex model (IM3)

Definition 6.8. *Open complex interaction model 2 (IM3)*

```
a( ensemble_classifier( TestExample, N, Ts, Result, Peers ), E ) ::
    a( meta_coordinator( TestExample, Ts, Peers, Answers ), M ) <- choose_peers( N, Peers ) then
    null <- vote( Answers, Result )

a( meta_coordinator( TestExample, Ts, Peers, Answers ), M ) then
    (
    a( coordinator( TestExample, PPeers, PAnswers ), R ) <-
        ( Ts = [ T | RestTs ] and pick_peers( T, Peers, PPeers, RestPeers ) ) then
    a( meta_coordinator( TestExample, RestTs, RestPeers, Answers ), M ) <-
        vote( PAnswers, Result ) and Answers = [ Result | RestAnswers ]
    )
    or
    null <- Ts = []
```

Definition 6.8 is another example of an open complex IM obtained through modification of IM2. `Ts` is a list of numbers. Each number indicates how many classifiers will be picked by `pick_peers`. The coordinator gets answers from different number of classifiers from the `Ts` set. This gives a different weight to each classifier's prediction.

6.2.4 Model for specificity metric (IM4)

Definition 6.9. *peer_ranker role for specificity metric (IM4)*

```
a( peer_ranker( QueryExample, QClass, N ), K ) ::
    a( ensemble_classifier( QueryExample, N, Result, Peers ), E ) then
    (
    // case of true negative
    null <- QClass = false and Result = false then
    null <- update( false, false, Peers )
    )
    or
    (
    // case of false positive
    null <- QClass = false and Result = true then
    null <- update( false, true, Peers )
    )
```

Definition 6.9 is a specified interaction model for the performance metric of specificity. The specificity metric is defined in the section of 7.6.2.4.

`peer_ranker` reflects the prediction result from `ensemble_classifier` differently based on the predicted class (`QClass`) and the actual class (`Result`). In the case of true negatives, it updates the result positively. In the case of false positives, it updates the result negatively. For other cases, it does not report the results to the peer ranking service because the value of specificity is determined only by the number of true negatives and false positives following its definition.

6.2.5 Model for high true positive rate and low false positive rate metrics (IM5)

Definition 6.10. *peer_ranker role for high TPR and low FPR metrics (IM5)*

```
a( peer_ranker( QueryExample, QClass, N ), K ) ::
    a( ensemble_classifier( QueryExample, N, Result, Peers ), E ) then
    (
    // case of true positive
    null <- QClass = true and Result = true then
    null <- update( true, true, Peers )
    )
    or
    (
    // case of false positive
    null <- QClass = false and Result = true then
    null <- update( false, true, Peers )
    )
```

Definition 6.10 is for the higher true positive rate (TPR) and the lower false positive rate (FPR). In the section 7.7.2.4, TPR and FPR are defined.

`peer_ranker` updates the prediction result from `ensemble_classifier` positively when the actual class is the positive and the predicted class is also the positive (true positives). On the other hand, it updates the result negatively when the actual class is the negative and the predicted class is the positive (false positives). For other cases, it does not reflect the prediction results.

6.2.6 Model with time constraint (IM6)

Definition 6.11. *peer_ranker role for time cost (IM6)*

```
a( peer_ranker( QueryExample, QClass, Time, N ), K ) ::
    null <- get_time( T1 ) then
    a( ensemble_classifier( QueryExample, N, Result, Peers ), E ) then
    null <- get_time( T2 ) then
    (
    null <- T2 - T1 =< Time then
    null <- update( Result, QClass, Peers )
    )
    or
    null <- update( true, false, Peers )
```

Prediction performance is the most fundamental evaluation metric on machine learning including ensemble learning. In addition to this, time cost is also a useful performance criterion.

Definition 6.11 is an open interaction model for time cost by re-writing the `peer_ranker` role. `Time` is the expected time in which we want an `ensemble_classifier` to give a prediction answer. If the actual time cost of an `ensemble_classifier` is higher than the expected cost, `peer_ranker` always updates the reputation of `Peers` negatively. When the actual cost is lower, `peer_ranker` applies the same constraint of `update( Result, QClass, Peers )` that is used in all the other IMs above. `Result` is a voted answer from the chosen peers (`Peers`) and `QClass` is the actual class.

6.3 Summary

All the open IMs shown so far are only some of the examples which can be written using LCC. IMs can be freely built by modifying existing roles, adding new roles, changing parameters, implementing other constraints or the other language elements of LCC.

Chapter 7

Experiments

7.1 General methodology

7.1.1 Binary class data

We only consider binary class data sets for our experiments. Binary class data is easier to measure and analyse than data for multi-class problems. In addition, some machine learning algorithms such as SVM [23] and boosting cannot easily address multi-class problems.

7.1.2 Training, query and test examples

We split each data set into three sets of examples for training, query and test. Training and test examples have the same role as in traditional machine learning. Query examples are for the peer ranking interaction that allows each peer to assess the reputation of other peers. In machine learning experiments, training and test examples are typically split by about a 9:1 ratio. Learners are trained sufficiently before testing. Query examples also have the same relationship for splitting. Learners needs to be trained sufficiently in querying. We separated each whole data set into training, query and test examples by 9:0.5:0.5 ratio respectively.

7.1.3 Base classifiers for pool preparation

We used 8 machine learning algorithms as templates for generating pool members (Table 7.1). Each pool member needs to have a different training status for diversity. We achieved this by

setting different initial points on learning with random number seeds. The 8 learner templates are all ensemble learning algorithms. We set the number of base classifiers as one for all of them (original default: 10). The templates are then no longer ensemble methods (they are simple learners as J-model takes the responsibility for coordinating the ensemble).

Through the two modifications, we can fairly compare J-model's performance with other ensemble methods. Detailed learning parameters are also described in Table 7.1. Weka [44], a machine learning framework, was used for pool preparation.

7.1.4 Static and dynamic conditions

J-model can do its interaction both under static and dynamic service conditions. Under static conditions, the classifier pool status does not change during interactions. Dynamic conditions permit several statuses such as classifier attending, missing or updating. We set 25% randomly selected classifiers from the pool to miss in every interaction for our experiments. Making a portion of classifiers miss is easier for conducting experiments under a controlled condition than taking attendance or update of classifiers. In addition, missing classifiers can give a more direct effect for dynamicity of classifiers. The portion of 25% was estimated as a reasonable level of this effect (this being a reasonably challenging number of missing classifiers that we would hope not to exceed in practice).

7.2 System implementation for experiments

We explain how our entire implemented system is assembled together for experiments from a classifier pool to an LCC interpreter.

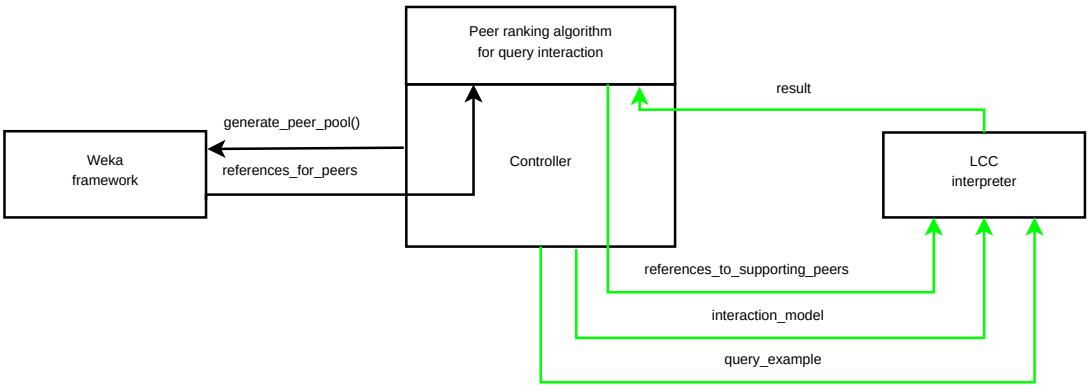

Figure 7.1: A system implementation for query interaction

Figure 7.1 shows the system implementation. The system is composed of three components; the Weka machine learning framework, a controller and an LCC interpreter. The Weka framework is being developed at the University of Waikato and an LCC interpreter named LiJ[1] is being developed by Nikolaos Chatzinikolaou. Both of the components are implemented with Java programming language. We implemented a controller to control the learning process and wrap the two other components and JRuby programming language was used to call Java classes natively. The peer ranking algorithm is an object in the controller component.

For query interactions, initially, the controller calls Weka to make Weka to generate a pool of classifiers. Weka returns references to generated peers to the the peer ranking algorithm through the controller. The controller calls the LCC interpreter with parameters of references to supporting peers (recommended peers), the definition of an interaction model and a query example. The LCC interpreter returns a prediction result to the controller and the controller applies the result to the peer ranking algorithm. Calling the LCC interpreter occurs multiple times according to the number of interactions we set.

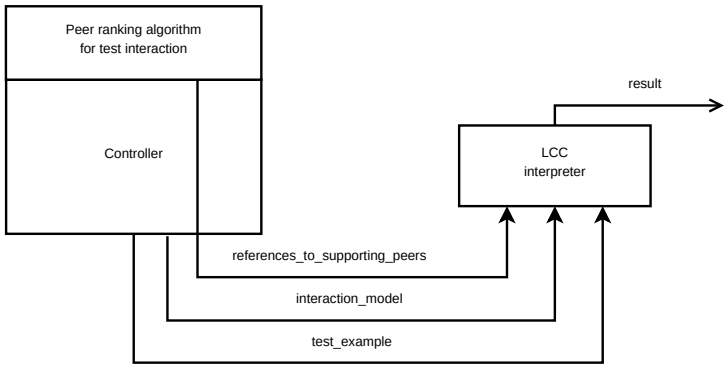

Figure 7.2: A system implementation for test interaction

Figure 7.2 shows how the system works for a test interaction. For tests, the controller and the LCC interpreter are used. The controller calls the LCC interpreter with parameters of references to recommended peers, the definition of an interaction model and a test example. The LCC interpreter gives a prediction result. We calls the LCC interpreter multiple times according to the number of test examples we have.

[1] http://sourceforge.net/projects/lij/

7.3 Peer separation from the pool

7.3.1 Introduction

We wish to know the extent to which better peers are separated from the other peers in the pool by interactions. Better peers are more appropriate ones for specific interaction results. The experiment of peer separation was accomplished with three sub experiments. First, we traced how many selections each peer get over interactions in the section 7.3.3.1. Second, we confirmed that more selected peers are higher ranked ones in the section 7.3.3.2. The peer ranking algorithm is designed to give more weight to better peers. Third, we investigated how much the gap is between higher ranked ones of 20% and lower ranked ones of 80% in the section 7.3.3.3. Experiments are executed under the static conditions and dynamic conditions described in Section 7.1.4.

7.3.2 Experimental setup

7.3.2.1 Data sets

Table 7.2: breast-cancer, kr-vs-kp and labor data sets

Name	Instances	Attributes	Categorical (symbolic) attributes	Numerical attributes	Missing values	Classes	Class ratio (majority class %)
breast-cancer	286	9	9	0	Yes	2	201:85 (70.28)
kr-vs-kp	3196	36	36	0	No	2	1669:1527 (55.22)
labor	57	16	8	8	Yes	2	20:37 (64.91)

We showed the result with the three standard data sets in Table 7.2. Each data set varies from the other standard benchmark data sets according to the properties given in the table.

7.3.2.2 Pool size and ensemble size

We experimented with pool sizes of 8, 16 and 32 (these being representative of the range of pool sizes we might expect to find in practice). Ensemble sizes are determined following the usual ratio of higher ranked peers which is up to 20%. So the ensemble size of 2 in the pool size of 8; 2 and 3 in 16; 2, 3 and 6 in 32 were chosen.

7.3.2.3 Choice of IM

Interaction model 1 (Definition 6.6) was used.

7.3.2.4 Number of interactions

We ran each experiment for 300 interactions (this providing what we assumed to be a reasonable length of time in which to construct ensembles via interaction)

7.3.3 Results under static conditions

7.3.3.1 Number of peers being selected over interactions

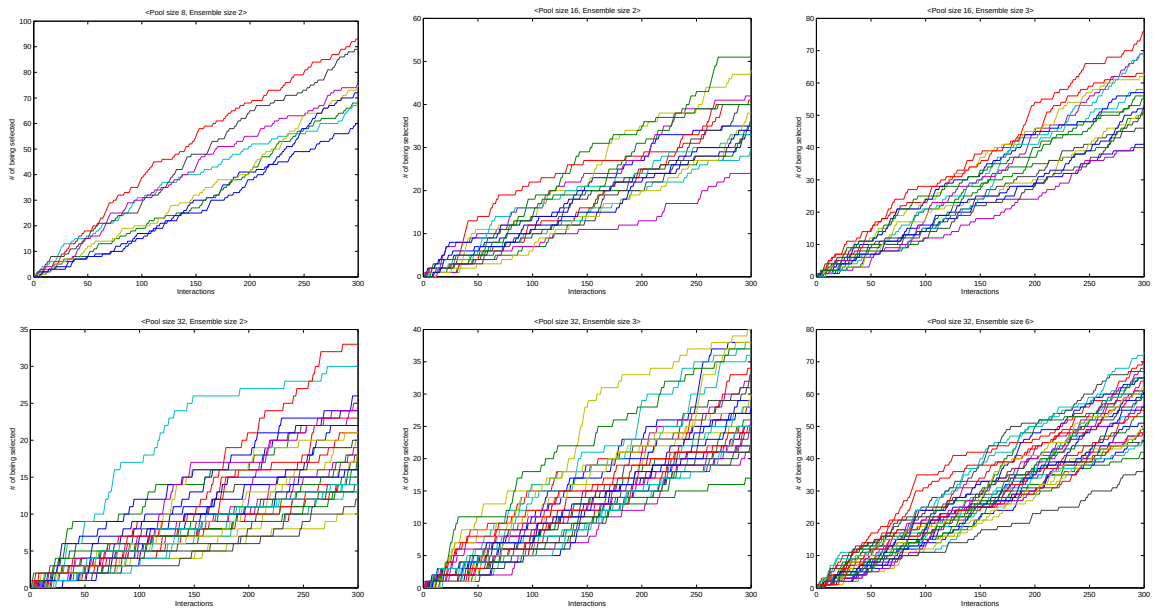

Figure 7.3: Number of peers being selected over interactions with breast-cancer under static conditions

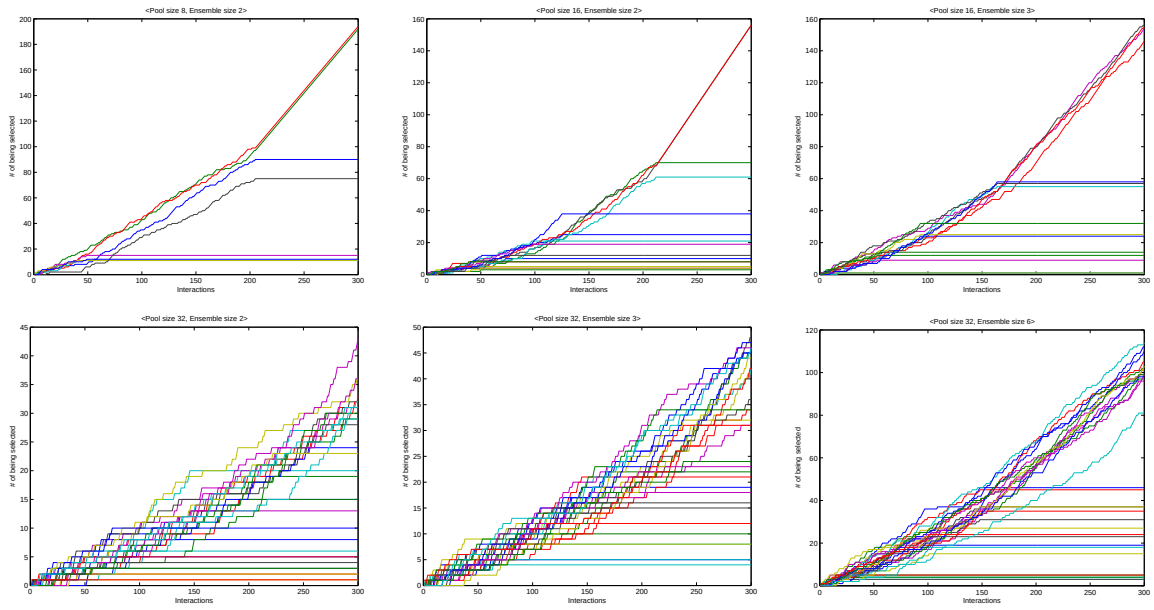

Figure 7.4: Number of peers being selected over interactions with kr-vs-kp under static conditions

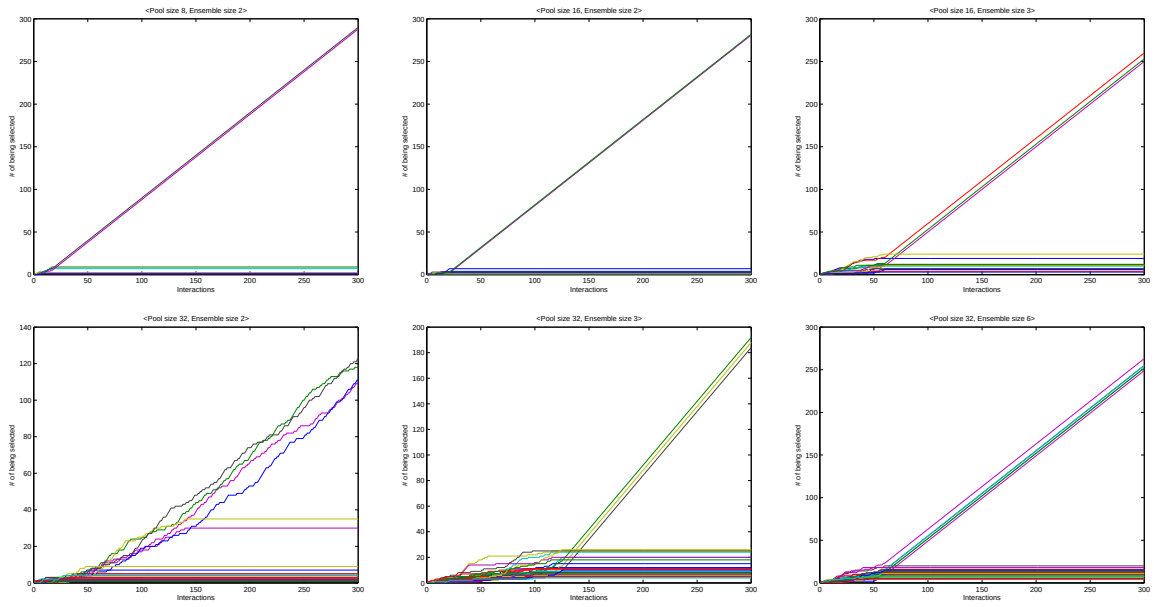

Figure 7.5: Number of peers being selected over interactions with labor under static conditions

Figures 7.3, 7.4 and 7.5 show the change of peers on how many they are selected by the peer ranking algorithm over interactions under static conditions. Different coloured lines in these graphs are different peers. Each data set showed somewhat different separation behaviours.

In breast-cancer, the amount of separation between peers in the early stage remains relatively

constant with gaps between the peers becoming gradually bigger over more interactions. kr-vs-kp showed a weeding-out of peers. While the interaction proceeds, some peers are cut from the selection (these are the horizontal lines in the graphs). labor showed an extreme case of separation. More selected peers in the early stage suppress all the other peers. This means that only the more selected ones are selected continually and the other ones do not have a chance to be selected. This separation among the peers is fixed at a very early stage.

7.3.3.2 Score over interactions

In the previous section 7.3.3.1, we gave the total number of peers being selected on each peer without considering plus-scored and minus-scored counts separately. In this section, we consider these values separately by calculating values of $plus\_counts - minus\_counts$.

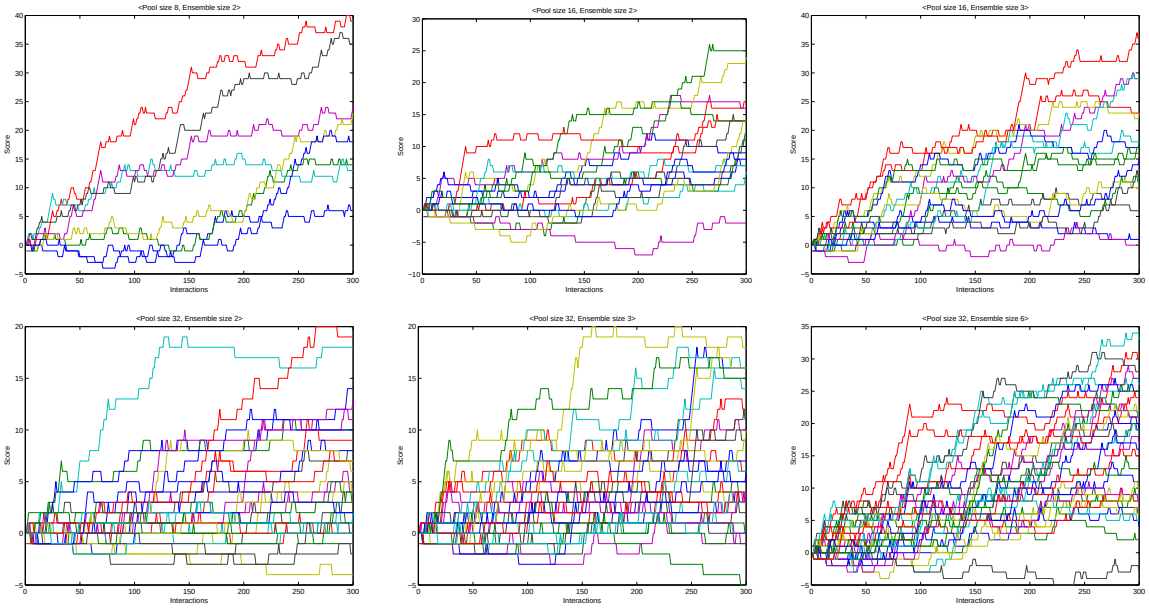

Figure 7.6: Score over interactions with breast-cancer under static conditions

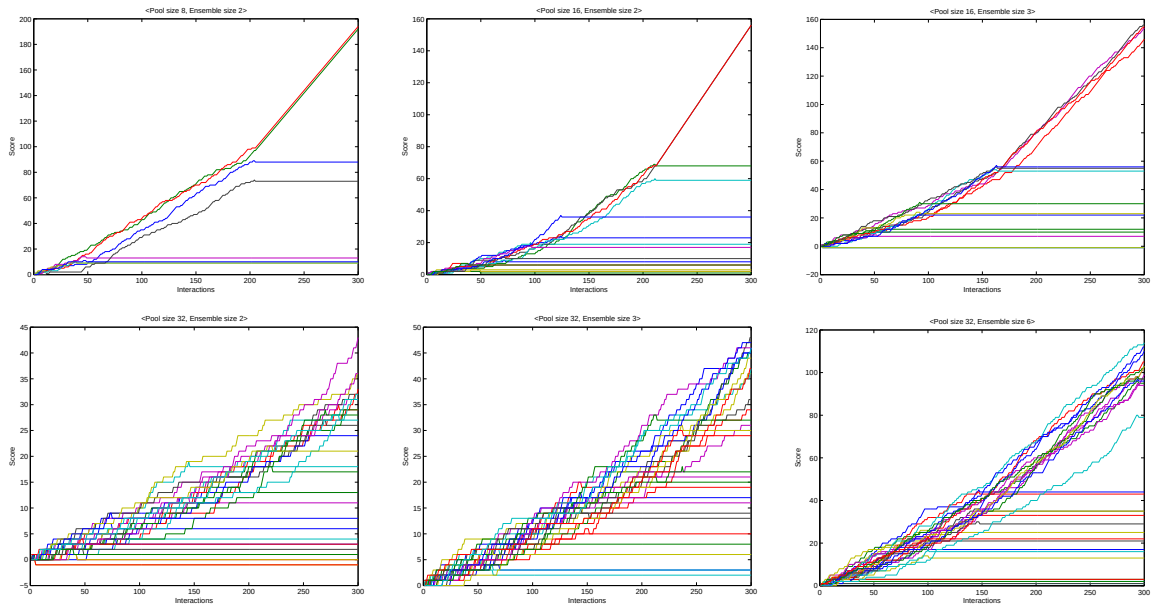

Figure 7.7: Score over interactions with kr-vs-kp under static conditions

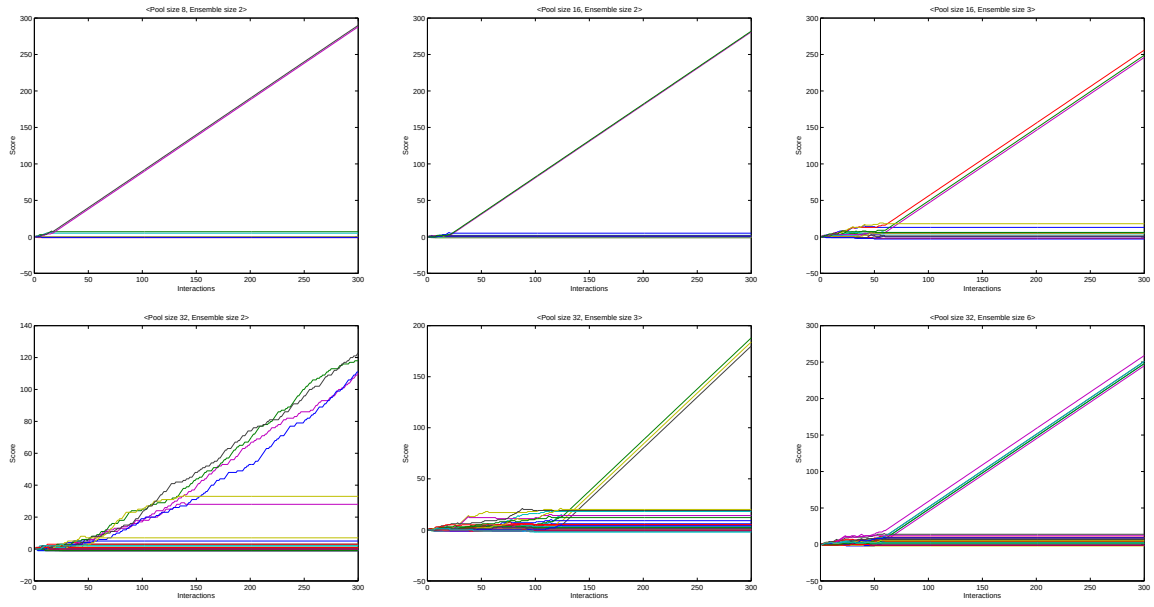

Figure 7.8: Score over interactions with labor under static conditions

The results in all the data sets above show that more frequently selected peers have more plus scores than the others. The ranking order of each peer in the figures here completely matchs with the orders in Figure 7.3, 7.4 and 7.5 above. This confirms that more selected peers actually get higher ranking.

7.3.3.3 Average scores of higher 20% and lower 80% scored peers over interactions

This experiment explores how much the gap changes between higher scored peers and lower scored peers over interactions.

Figure 7.9: Average scores interactions with breast-cancer under static conditions

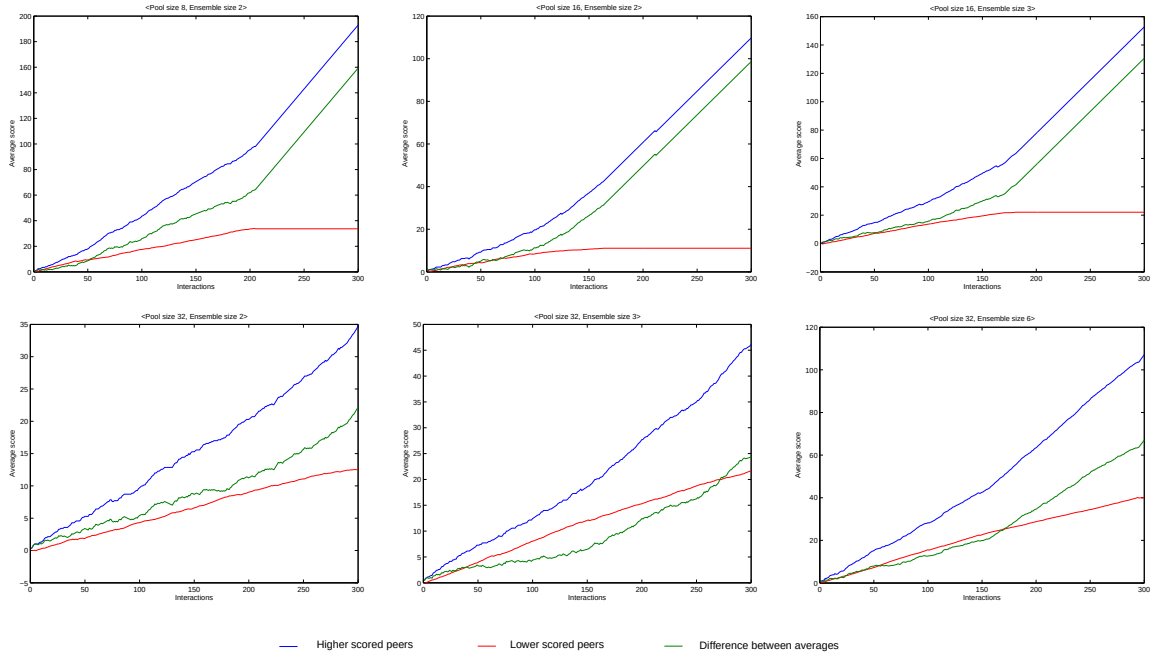

Figure 7.10: Average scores interactions with kr-vs-kp under static conditions

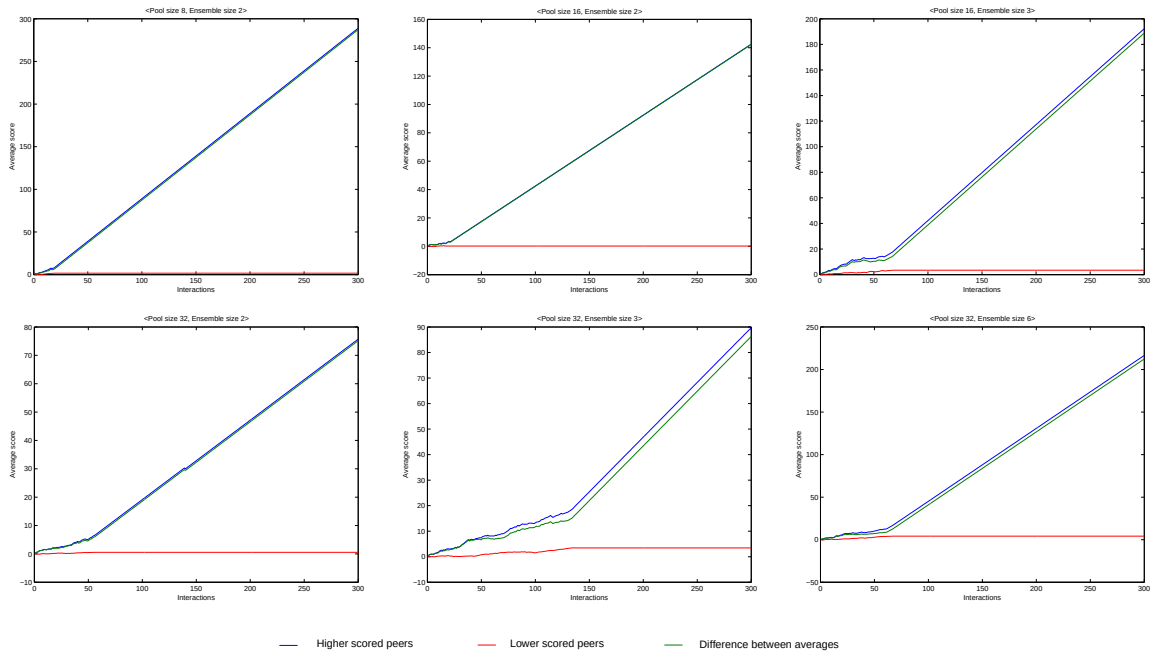

Figure 7.11: Average scores interactions with labor under static conditions

In breast-cancer, the blue line (the average score of 20% higher ranked peers) and the red line (the average score of 80% lower ranked peers) ascend with increasing interactions. The gap between them gradually increases at the same time. We can see that the green line (difference between averages) ascends over interactions. The gradient of the green line, however, becomes flatter after some point during interaction. In kr-vs-kp, the blue and red lines also ascend with increasing interactions. However the gradient of the blue line becomes steeper after some point while the red line becomes flatter. The gap becomes bigger over interactions. labor showed an extreme case. 80% lower scored peers hardly got any scores. 20% higher scored peers got almost all scores.

7.3.4 Results under dynamic conditions

7.3.4.1 Number of peers being selected over interactions

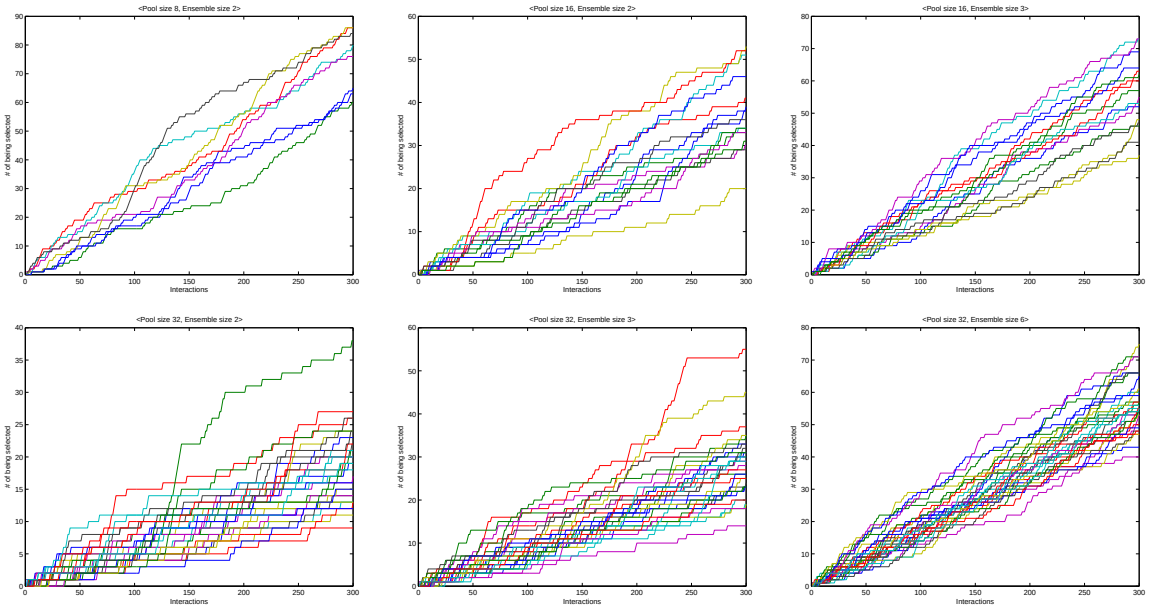

Figure 7.12: Number of peers being selected over interactions with breast-cancer under dynamic conditions

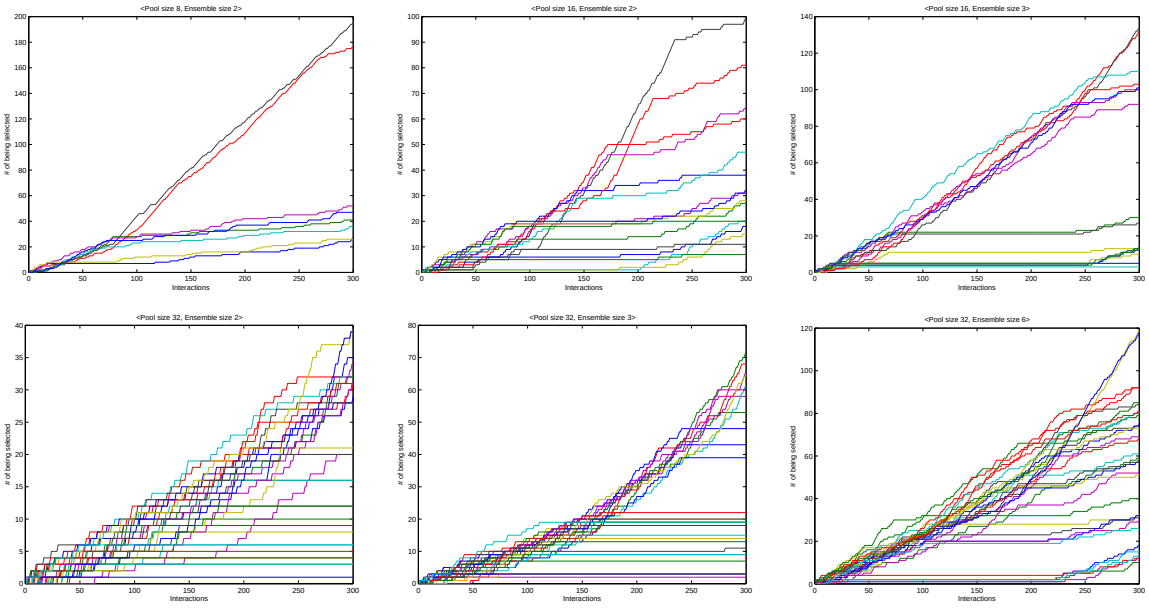

Figure 7.13: Number of peers being selected over interactions with kr-vs-kp under dynamic conditions

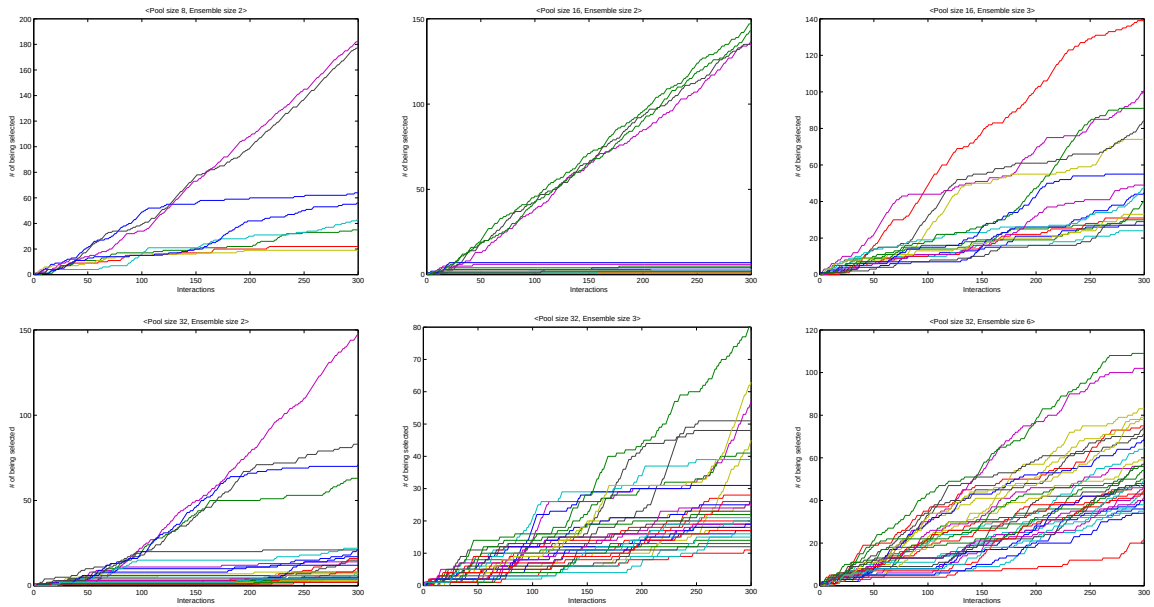

Figure 7.14: Number of peers being selected over interactions with labor under dynamic conditions

Satisfactory peer separation occurred all over the data sets and various pool/ensemble sizes like the separation under the static condition. It means that J-model is robust for peer separation under dynamic conditions.

Additionally, there is a difference compared with the results of the static conditions. As remarked in the cases of kr-vs-kp and labor, the lower scored peers that were certainly suppressed under static conditions shown in Figure 7.4 and 7.5 (they had much lower scores than the higher scored peers had) are less suppressed under dynamic conditions (they have more opportunity to get scores). This shows that the higher scored peers have less opportunity to get scores under dynamic conditions as well.

7.3.4.2 Score over interactions

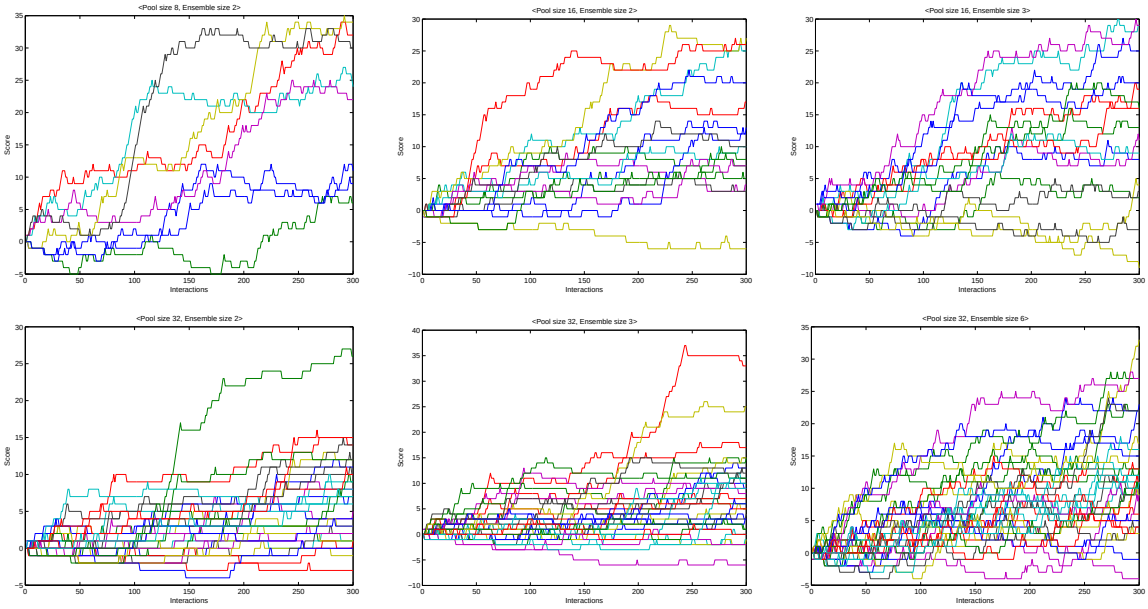

Figure 7.15: Score over interactions with breast-cancer under dynamic conditions

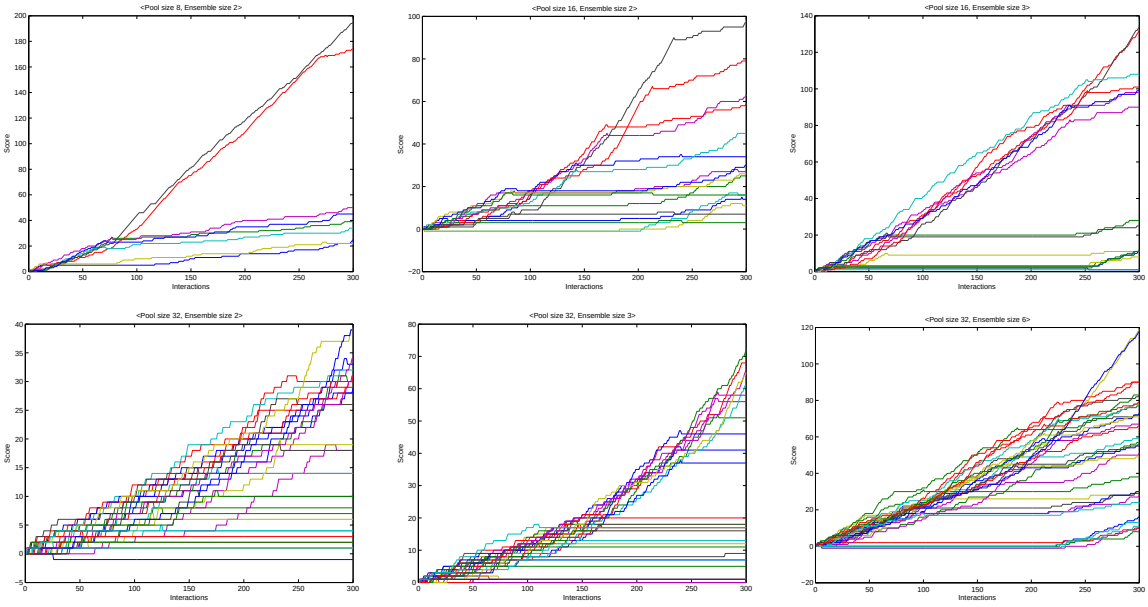

Figure 7.16: Score over interactions with kr-vs-kp under dynamic conditions

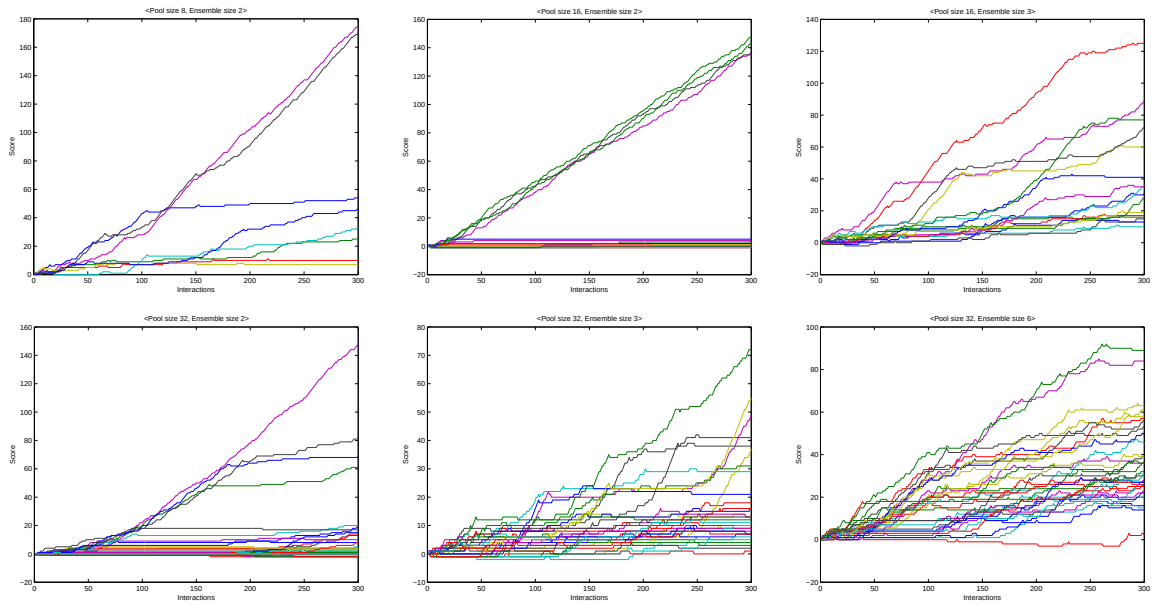

Figure 7.17: Score over interactions with labor under dynamic conditions

The ranking orders matched with the peer orders shown in Section 7.3.4.1.

7.3.4.3 Average scores of higher 20% and lower 80% scored peers over interactions

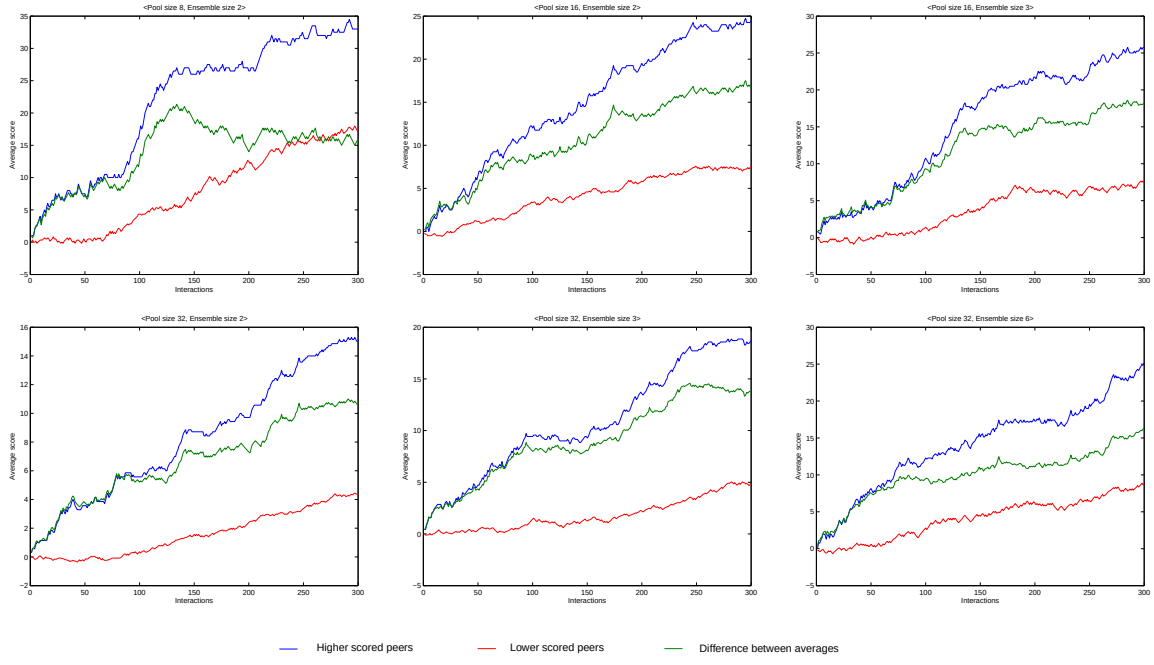

Figure 7.18: Average scores interactions with breast-cancer under dynamic conditions

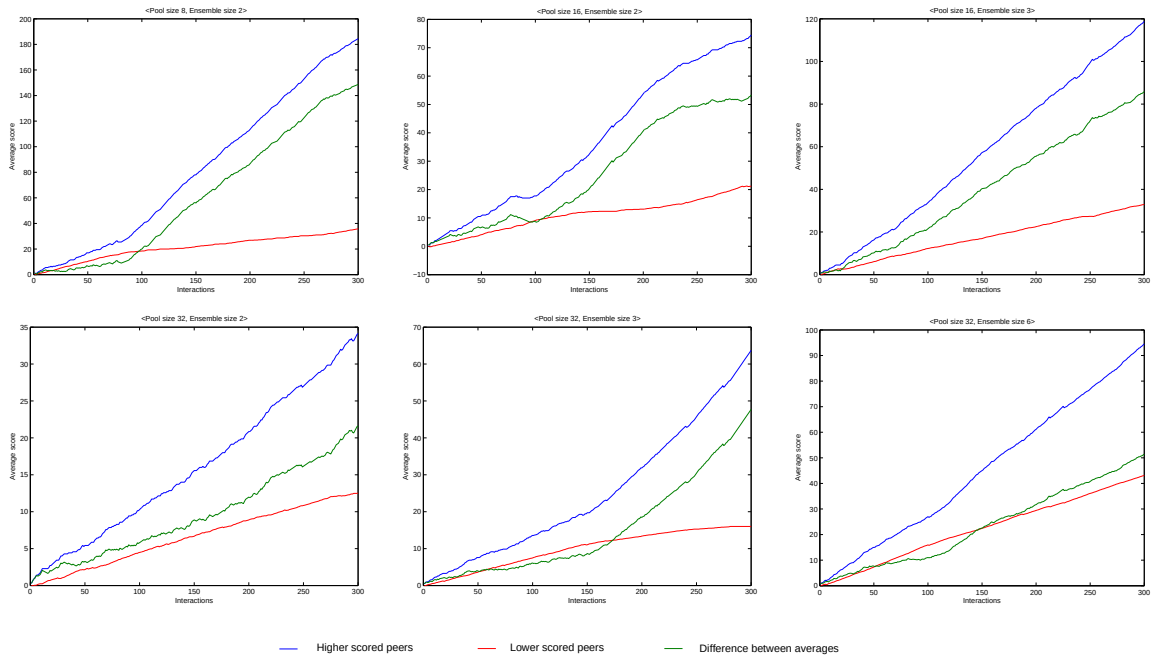

Figure 7.19: Average scores interactions with kr-vs-kp under dynamic conditions

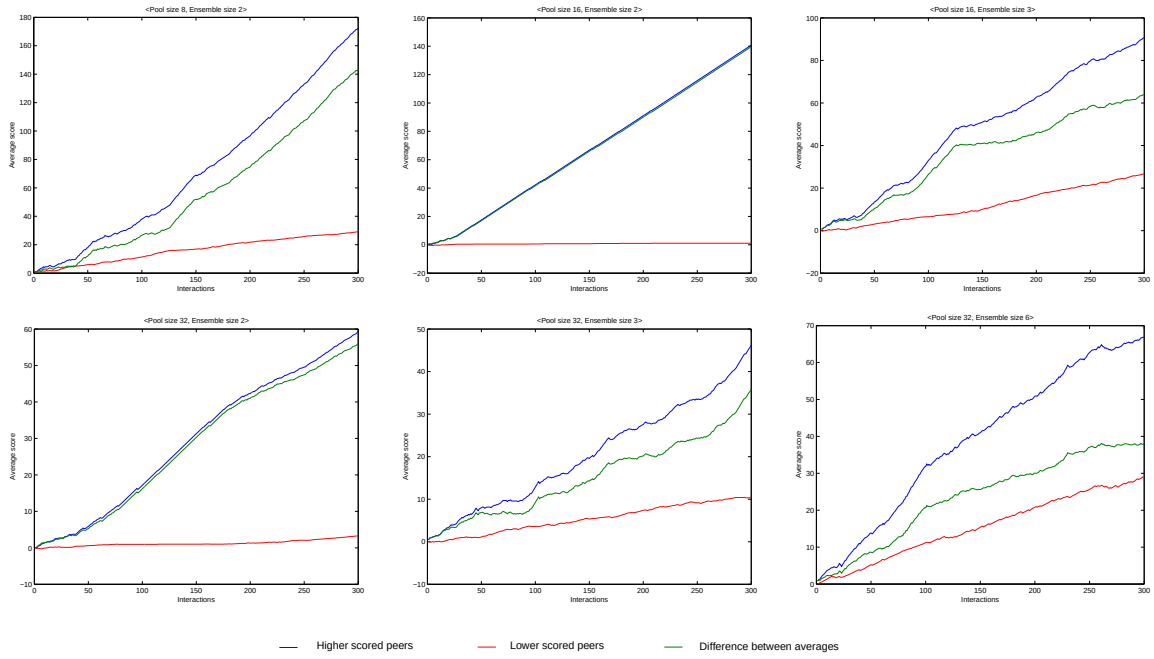

Figure 7.20: Average scores interactions with labor under dynamic conditions

Overall, the gradients of increase are less sharp than the gradients of the static results. The difference gaps, however, increased.

7.4 Peer convergence to optima

7.4.1 Introduction

We showed that the participating peers are separated based on their scores over interactions in Section 7.3. The higher scored ones compose the ensemble. In this section, we investigated how common higher scored peers would be selected after repeated trials using different initial selections of peers. This confirms that converged peer groups (local or global optimas) can be obtained from different starting points.

7.4.2 Experimental setup

We fixed the number of interactions at 200 (which we expected to be enough for convergence). This value is based on the results of the number of peers being selected and score over interactions in Section 7.3. In those results, the separation become mature at least after 200 interactions.

We set 20, 40, 60, 80 and 100 trials for repetition. Each trial starts from different peers which were randomly chosen.

7.4.3 Results under static conditions

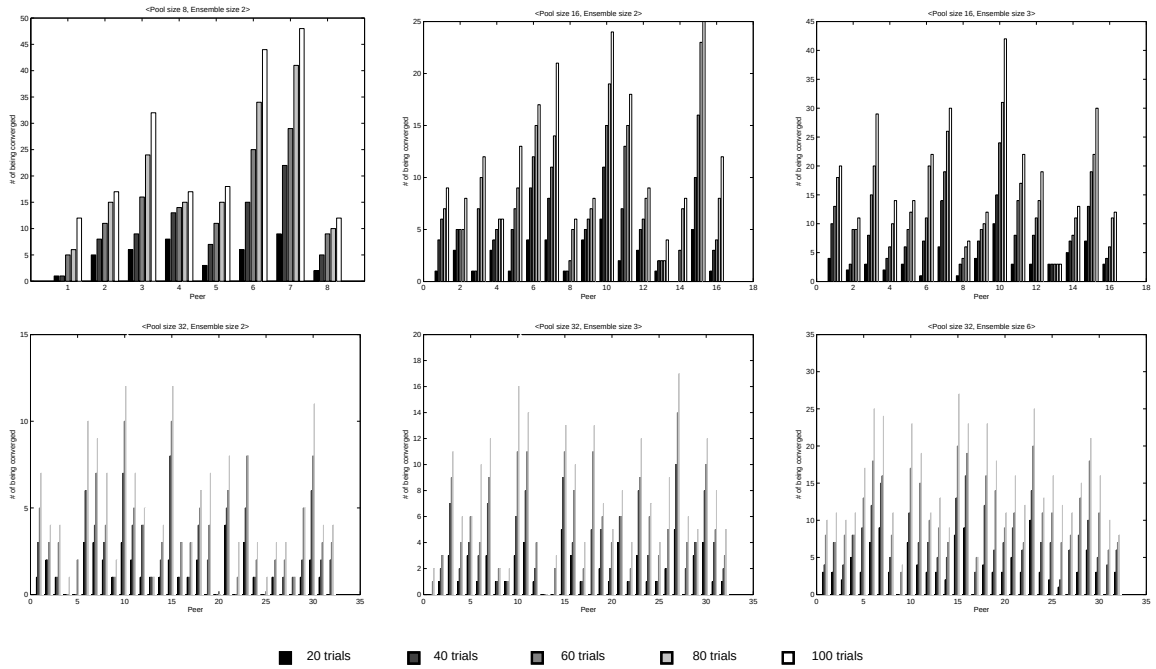

Figure 7.21: Number of peers being converged at 200 interactions with breast-cancer under static conditions

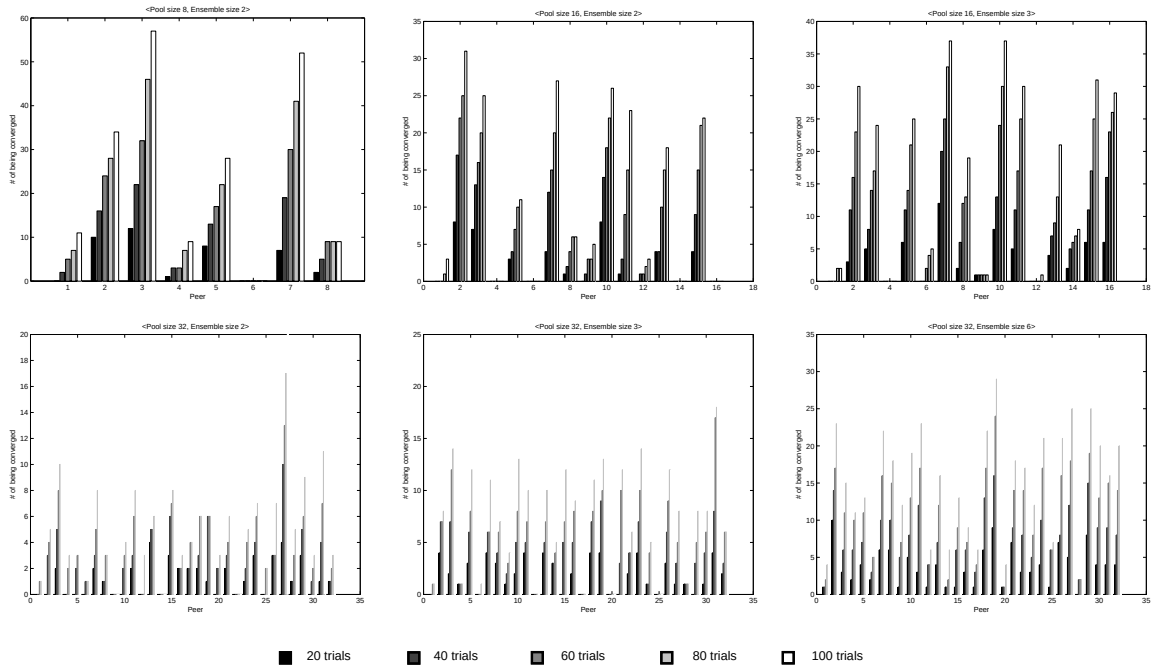

Figure 7.22: Number of peers being converged at 200 interactions with kr-vs-kp under static conditions

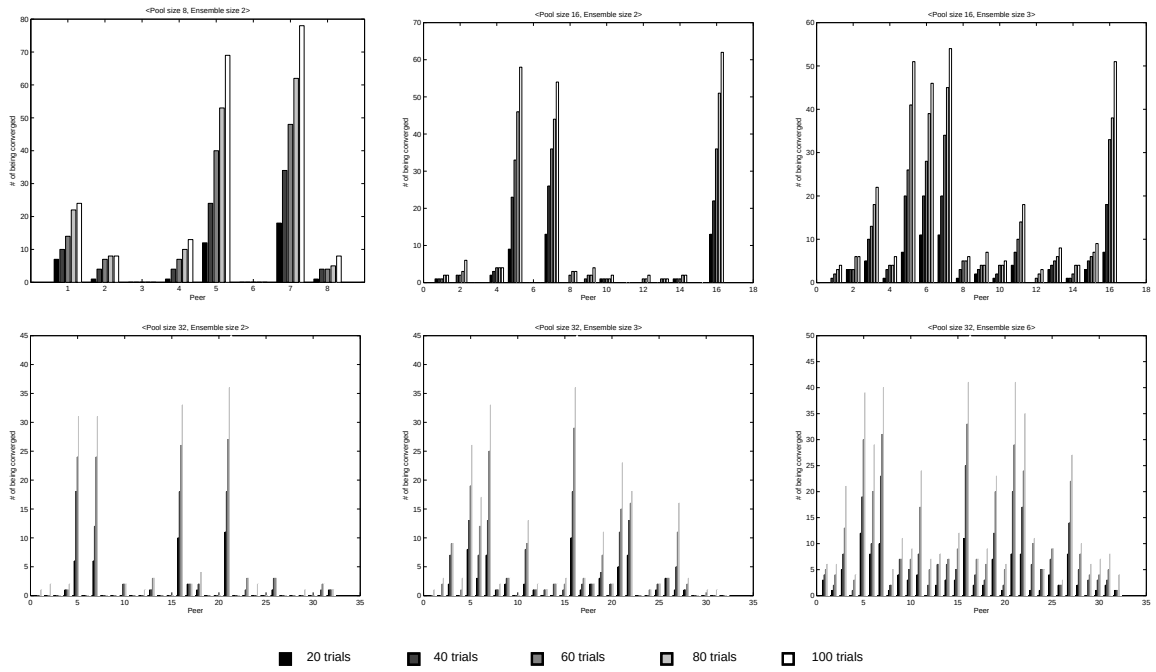

Figure 7.23: Number of peers being converged at 200 interactions with labor under static conditions

Each peer had a different frequency of being converged. For example, in the results of breast-cancer; pool size 8; ensemble size 2, peers 2, 3, 4, 6 and 7 were more frequently converged than the other peers on 20 trials. On 100 trials, peers 3, 6 and 7 finally got almost all selections.

The results showed two features of the convergence of peers.

- Fixation - The peers who have greater frequency on a small number of trials relatively match with the peers who have greater frequency on a larger number of trials. This means that more frequently converged peers on a small number of trials might also be more frequently converged ones on a larger number of trials. For example, in Figure 7.23, peer 5 and 7 show this tendency.
- Acceleration - The number being converged is accelerated with more trials. Those peers that have more frequency get more frequency.

7.4.4 Results under dynamic conditions

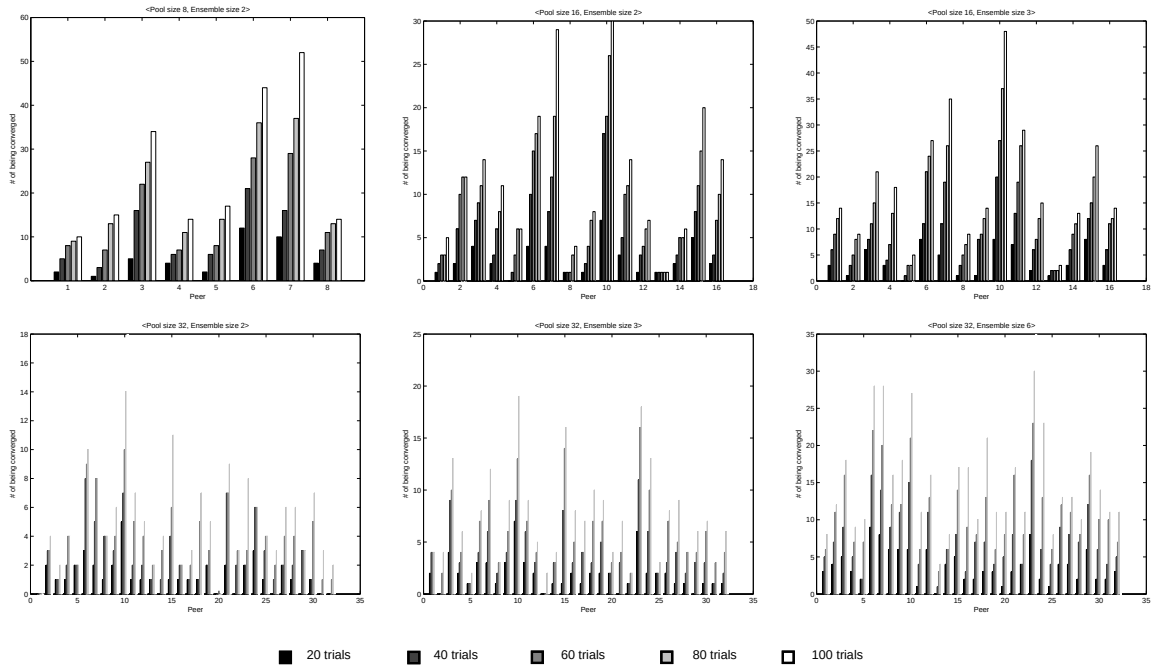

Figure 7.24: Number of peers being converged at 200 interactions with breast-cancer under dynamic conditions

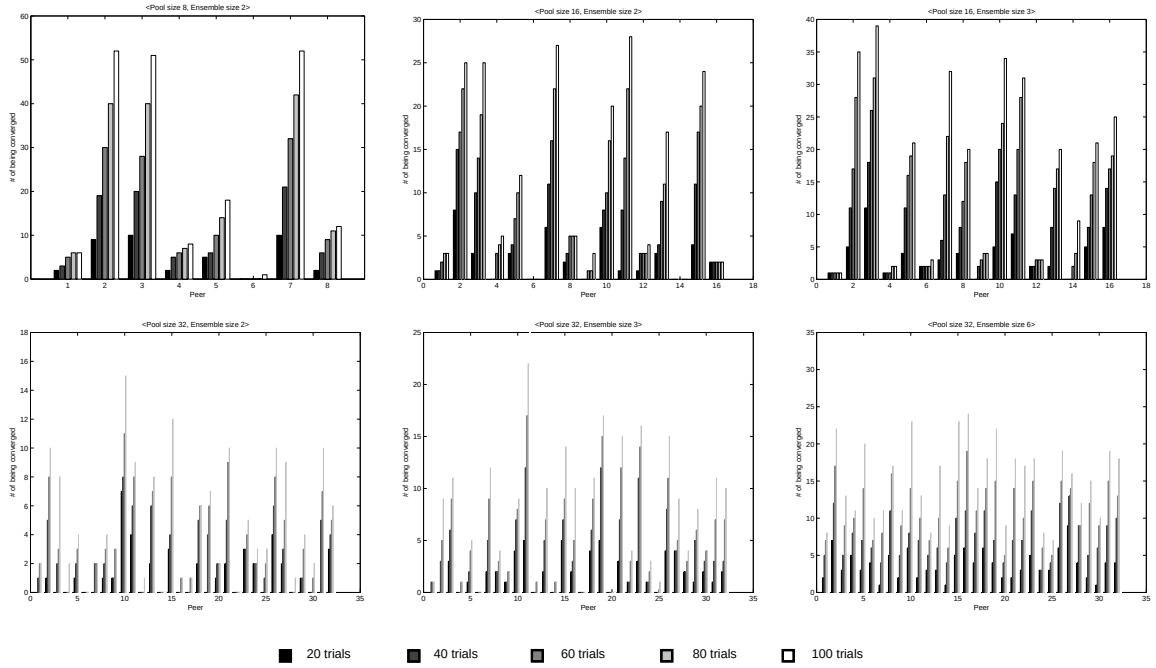

Figure 7.25: Number of peers being converged at 200 interactions with kr-vs-kp under dynamic conditions

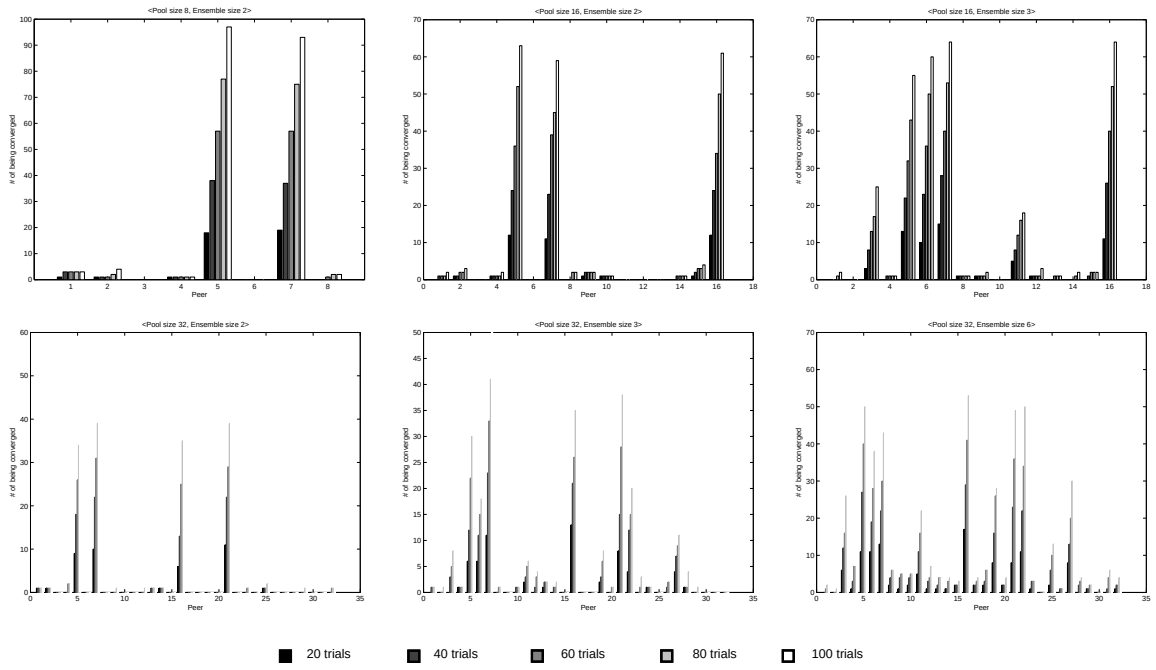

Figure 7.26: Number of peers being converged at 200 interactions with labor under dynamic conditions

Under dynamic conditions, the results of convergence showed the same features as the results under static conditions.

7.4.5 Conclusion

The experiment on the peer separation showed that J-model separates peers which provide the higher success rate than the other peers from the peer pool when taking roles in the interaction model of of Definition 6.6. The other experiment on peer convergence showed that the separated peers persistently converge to optima on several trials.

7.5 Learning curves

7.5.1 Introduction

Learning curves help our understanding of a connection between an ensemble of selected peers and its ability for classification. A learning curve describes what classification performance

an ensemble gives over interactions. Through performance change in a learning curve, we can trace the classification ability of an ensemble.

7.5.2 Experimental setup and methods

We composed an ensemble classifier of highly scored peers according to each ensemble size.

We traced the classification ability of an ensemble over 600 interactions. 600 interactions are sufficient to enable J-model to get to a stable phase. We measured values in every 10 interactions.

We showed the results about 10 sub-data sets for each experiment. A sub data set is one of the 10 folded data sets of the original data set.

We performed these experiments of getting learning curves under static conditions. As shown in the previous experiments in Section 7.3, peers in a pool were separated well into higher scored and lower scored peers under both of static and dynamic conditions and higher scored peers also converged to global optimas under both of the conditions in Section 7.4. For this reason, experiments under static conditions remove the need for extra experiments under dynamic conditions.

We showed representative results with three data sets of breast-cancer, kr-vs-kp and labor. Among different 13 benchmark data sets, breast-cancer, kr-vs-kp and labor gave patterns of a learning curve respectively.

7.5.3 Results

The results can be analysed according to several viewpoints on the sequences of interactions: the initial status, the early steps, the middle and latter steps and the change through them.

7.5.3.1 breast-cancer

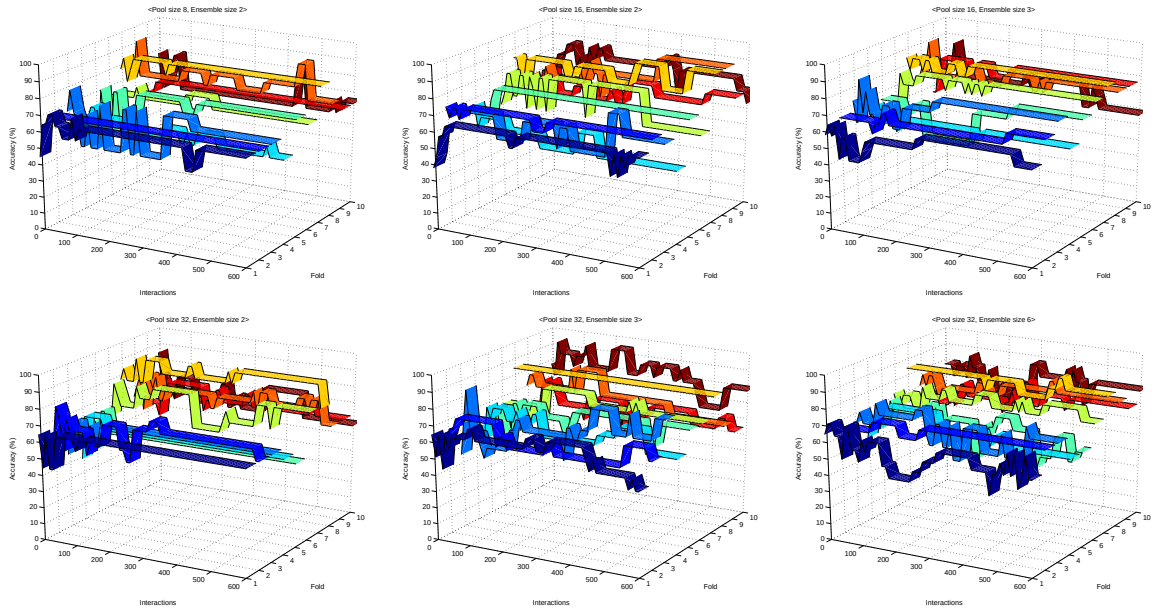

Figure 7.27: Learning curves of breast-cancer

- Initial - The experiment starts from random points.
- Early - There is fluctuation.
- Middle and latter - The change becomes more stable although there remains fluctuation. The fluctuation is less serious and frequent than it is in the early steps.
- Change - It is difficult to say precisely how the accuracy increases over interactions. There are, however, some ensembles that maintain their higher accuracies.

7.5.3.2 kr-vs-kp

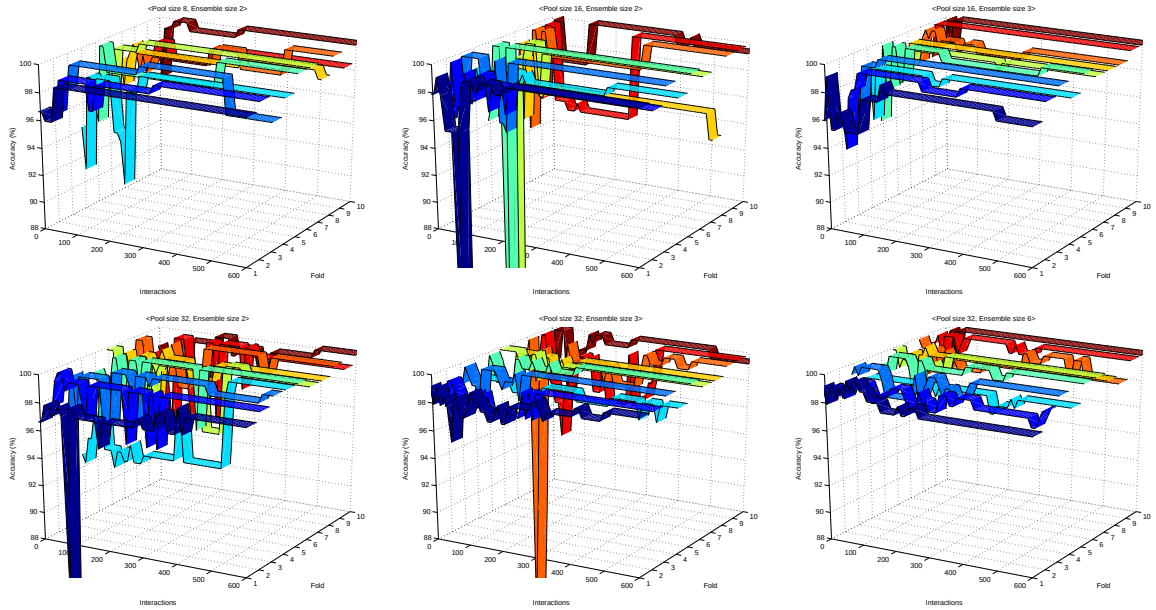

Figure 7.28: Learning curves of kr-vs-kp

- Initial - The experiment starts from random points.
- Early - There is serious fluctuation.
- Middle and latter - The change becomes stable.
- Change - The accuracy increases over interactions and almost all ensembles keep their higher accuracies.

7.5.3.3 labor

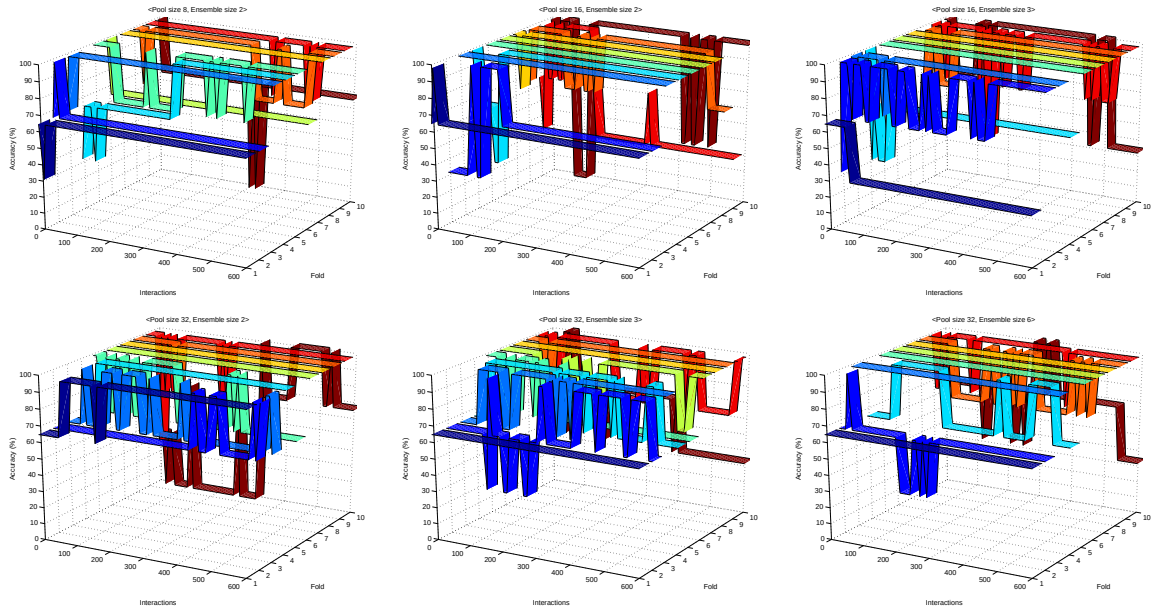

Figure 7.29: Learning curves of labor

- Initial - The experiment starts from random points.
- Early - There is fluctuation.
- Middle and latter - The change becomes stable or circulating.
- Change - Many ensembles keep their higher accuracies and some ensembles show cycle changes.

7.5.4 Conclusion

The accuracy changes showed somewhat different aspects for different data sets. But they have common features. First, they become more stable over interactions. Second, they retain their higher accuracies over interactions.

7.6 Benchmark comparisons

7.6.1 Introduction

We compared J-model's classification performance with other representative ensemble methods including AdaBoost, Bagging, Decorate [63], LogitBoost [36], RandomCommittee [58], RandomForest [12], RandomSubSpace and RotationForest [81].

7.6.2 Experimental setup

7.6.2.1 Benchmark data sets

Table 7.3: Benchmark data sets

Index	Name	Instances	Attributes	Categorical (symbolic) attributes	Numerical attributes	Missing values	Classes	Class ratio (majority class %)
1	breast-cancer	286	9	9	0	Yes	2	201:85 (70.28)
2	breast-w	699	9	0	9	Yes	2	458:241 (65.52)
3	credit-a	690	15	9	6	Yes	2	307:383 (55.51)
4	credit-g	1000	20	13	7	No	2	700:300 (70.00)
5	diabetes	768	8	0	8	No	2	500:268 (65.10)
6	heart-statlog	270	13	0	13	No	2	150:120 (55.56)
7	hepatitis	155	19	13	6	Yes	2	32:123 (79.35)
8	ionosphere	351	34	34	0	No	2	126:225 (64.10)
9	kr-vs-kp	3196	36	36	0	No	2	1669:1527 (55.22)
10	labor	57	16	8	8	Yes	2	20:37 (64.91)
11	mushroom	8124	22	22	0	Yes	2	4208:3916 (51.80)
12	sonar	208	60	0	60	No	2	97:111 (53.37)
13	vote	435	16	16	0	Yes	2	267:168 (61.38)

The 13 benchmark data sets [34] were selected based on the variety of their properties and considering the size. All data sets are binary class problems.

7.6.2.2 J-model settings

Pool preparation

- Base classifiers - We generated members of the pool using the classifier templates defined in Table 7.1.

- Pool size - We set the pool size as 64 for comparison experiments.

Ensemble size We set the ensemble size as 10. 10 is about 20% of the pool size 64.

Interaction models We used the interaction models of IM1, IM2, IM3, IM4 and IM6 defined in Chapter 6.

Number of interactions We fixed the number of interactions as 200.

7.6.2.3 The other representative algorithms

We compared J-model's performance with other 8 traditional ensemble methods including AdaBoost, Bagging, Decorate, LogitBoost, RandomCommittee, RandomForest, RandomSubSpace and RotationForest. All the ensemble methods used 10 for their ensemble sizes.

7.6.2.4 Evaluation technique

10-fold cross validation We used 10-fold cross validation to generalise the analysis results to independent data sets.

Performance metrics

- *Accuracy* (ACC) - $(TP+TN)/(P+N)$
- Sensitivity or recall - $TP/P = TP/(TP+FN)$
- Specificity - $TN/N = TN/(FP+TN)$
- F-measure - $2 \times \frac{precision \times recall}{precision+recall}$. F-measure is a weighted average of the precision and recall.
- Area under curve (AUC) - AUC is the area under the ROC curve.
- Time - Time cost for training and testing. Time for training means how much time a model needs to be trained with a training data set. Time for test indicates how much time a trained model requires to classify test examples.

P: the positives, N: the negatives

TP: true positives, FP: false positives, TN: true negatives, FN: false negatives

Precision: $TP/(TP+FP)$. Precision is the fraction of retrieved instances that are relevant while recall is the fraction of relevant instances that are retrieved.

ROC: receiver operating characteristic. ROC curve is a graphical plot of the true positive rate (sensitivity) vs false positive rate (1 - specificity) for a binary classifier. ROC analysis provides tools to select possibly optimal models and is related in a direct and natural way to const/benefit analysis of diagnostic decision making.

7.6.3 Results

Table 7.4 shows the comparison results for all the benchmark data sets.

Table 7.4: Benchmark comparison results

Data set	Ensemble algorithm	Performance metrics					
		Accuracy (%)	Sensitivity	Specificity	F-measure	AUC	Time for training; time for test (seconds)
breast-cancer	J-model (IM1)	72.095 (11.158)	0.721 (0.112)	0.499 (0.151)	0.686 (0.123)	0.689 (0.169)	6.611; 0.078
	J-model (IM2)	70.714 (11.901)	0.707 (0.119)	0.483 (0.162)	0.668 (0.135)	0.688 (0.206)	-
	J-model (IM3)	67.905 (11.903)	0.679 (0.119)	0.436 (0.136)	0.641 (0.121)	0.682 (0.158)	-
	J-model (IM4)	67.952 (12.566)	0.679 (0.126)	0.450 (0.151)	0.642 (0.131)	0.638 (0.182)	-
	J-model (IM6)	68.048 (9.415)	0.680 (0.094)	0.425 (0.156)	0.626 (0.119)	0.656 (0.190)	6.624; 0.015
	AdaBoostM1	70.283 (9.165)	0.703 (0.092)	0.544 (0.135)	0.688 (0.091)	0.716 (0.117)	0.218; 0.005
	Bagging	68.879 (6.733)	0.689 (0.067)	0.386 (0.079)	0.634 (0.072)	0.649 (0.119)	0.540; 0.006
	Decorate	73.461 (7.370)	0.735 (0.074)	0.535 (0.108)	0.715 (0.080)	0.654 (0.120)	1.204; 0.002
	LogitBoost	72.401 (8.031)	0.724 (0.080)	0.518 (0.117)	0.700 (0.089)	0.694 (0.119)	0.341; 0.006
	RandomCommittee	67.562 (7.770)	0.676 (0.078)	0.452 (0.102)	0.651 (0.079)	0.633 (0.129)	0.248; 0.007
	RandomSubSpace	70.998 (3.993)	0.710 (0.040)	0.367 (0.078)	0.630 (0.058)	0.663 (0.110)	0.184; 0.016
	RotationForest	73.485 (7.432)	0.735 (0.074)	0.470 (0.106)	0.694 (0.086)	0.669 (0.120)	2.668; 0.051
	RandomForest	69.286 (6.325)	0.693 (0.063)	0.486 (0.098)	0.672 (0.067)	0.654 (0.093)	0.090; 0.002
breast-w	J-model (IM1)	96.572 (3.790)	0.966 (0.038)	0.962 (0.057)	0.966 (0.038)	0.995 (0.008)	3.859; 0.031
	J-model (IM2)	96.572 (3.332)	0.966 (0.033)	0.958 (0.056)	0.965 (0.034)	0.996 (0.004)	-
	J-model (IM3)	96.286 (4.247)	0.963 (0.042)	0.953 (0.066)	0.963 (0.043)	0.995 (0.007)	-
	J-model (IM4)	96.857 (3.928)	0.968 (0.039)	0.960 (0.056)	0.968 (0.040)	0.996 (0.006)	-
	J-model (IM6)	95.714 (4.651)	0.957 (0.046)	0.945 (0.066)	0.957 (0.047)	0.993 (0.012)	4.378; 0.000
	AdaBoost	94.849 (2.943)	0.948 (0.029)	0.936 (0.046)	0.948 (0.030)	0.989 (0.009)	0.232; 0.001
	Bagging	95.563 (2.963)	0.956 (0.030)	0.953 (0.041)	0.956 (0.030)	0.988 (0.011)	0.319; 0.002
	Decorate	95.704 (3.733)	0.957 (0.037)	0.946 (0.050)	0.957 (0.038)	0.990 (0.010)	3.195; 0.001
	LogitBoost	95.708 (2.020)	0.957 (0.020)	0.946 (0.026)	0.957 (0.020)	0.992 (0.007)	0.345; 0.000
	RandomCommittee	95.994 (2.540)	0.960 (0.025)	0.951 (0.047)	0.960 (0.026)	0.988 (0.013)	0.268; 0.004
	RandomSubSpace	94.849 (2.727)	0.948 (0.027)	0.947 (0.043)	0.949 (0.027)	0.982 (0.020)	0.249; 0.007
	RotationForest	97.137 (1.696)	0.971 (0.017)	0.975 (0.016)	0.972 (0.017)	0.988 (0.012)	1.519; 0.064
	RandomForest	96.137 (2.219)	0.961 (0.022)	0.958 (0.027)	0.961 (0.022)	0.987 (0.014)	0.281; 0.004
credit-a	J-model (IM1)	88.000 (4.572)	0.880 (0.046)	0.879 (0.046)	0.880 (0.046)	0.930 (0.035)	8.397; 0.031
	J-model (IM2)	87.714 (3.626)	0.877 (0.036)	0.876 (0.035)	0.877 (0.036)	0.939 (0.035)	-
	J-model (IM3)	87.714 (3.393)	0.877 (0.034)	0.875 (0.035)	0.875 (0.035)	0.930 (0.036)	-
	J-model (IM4)	89.429 (3.143)	0.894 (0.031)	0.891 (0.030)	0.894 (0.031)	0.932 (0.032)	-
	J-model (IM6)	87.143 (3.441)	0.871 (0.034)	0.874 (0.034)	0.871 (0.034)	0.938 (0.028)	8.630; 0.015
	AdaBoost	84.638 (2.913)	0.846 (0.029)	0.844 (0.025)	0.846 (0.029)	0.932 (0.022)	0.185; 0.000
	Bagging	84.928 (4.546)	0.849 (0.045)	0.851 (0.045)	0.850 (0.045)	0.914 (0.031)	0.570; 0.000
	Decorate	85.942 (3.433)	0.859 (0.034)	0.855 (0.038)	0.859 (0.034)	0.919 (0.025)	3.906; 0.004
	LogitBoost	84.928 (3.562)	0.849 (0.036)	0.852 (0.032)	0.849 (0.035)	0.936 (0.022)	0.273; 0.000
	RandomCommittee	83.478 (3.502)	0.835 (0.035)	0.830 (0.042)	0.834 (0.036)	0.899 (0.035)	0.379; 0.005
	RandomSubSpace	86.522 (4.895)	0.865 (0.049)	0.856 (0.052)	0.864 (0.049)	0.919 (0.034)	0.490; 0.004
	RotationForest	85.652 (3.333)	0.857 (0.033)	0.855 (0.032)	0.856 (0.033)	0.918 (0.030)	6.605; 0.125
	RandomForest	85.072 (3.726)	0.851 (0.037)	0.849 (0.036)	0.851 (0.037)	0.912 (0.031)	0.333; 0.003
credit-g	J-model (IM1)	73.200 (5.154)	0.732 (0.052)	0.531 (0.080)	0.712 (0.056)	0.769 (0.071)	19.358; 0.000
	J-model (IM2)	73.200 (4.833)	0.732 (0.048)	0.550 (0.080)	0.716 (0.051)	0.758 (0.077)	-

	J-model (IM3)	71.600 (6.248)	0.716 (0.062)	0.520 (0.094)	0.697 (0.066)	0.746 (0.080)	-
	J-model (IM4)	74.400 (6.800)	0.744 (0.068)	0.566 (0.089)	0.730 (0.069)	0.749 (0.089)	-
	J-model (IM6)	71.800 (5.618)	0.718 (0.056)	0.510 (0.081)	0.696 (0.057)	0.731 (0.088)	19.982; 0.015
	AdaBoost	69.500 (2.655)	0.695 (0.027)	0.452 (0.068)	0.662 (0.042)	0.725 (0.049)	0.316; 0.000
	Bagging	74.900 (4.826)	0.749 (0.048)	0.571 (0.077)	0.734 (0.051)	0.776 (0.058)	1.054; 0.001
	Decorate	72.900 (3.673)	0.729 (0.037)	0.562 (0.063)	0.717 (0.039)	0.737 (0.032)	6.838; 0.006
	LogitBoost	70.800 (4.045)	0.708 (0.040)	0.484 (0.057)	0.683 (0.042)	0.731 (0.054)	0.394; 0.001
	RandomCommittee	73.900 (4.784)	0.739 (0.048)	0.555 (0.078)	0.723 (0.053)	0.762 (0.068)	0.516; 0.009
	RandomSubSpace	73.800 (3.059)	0.738 (0.031)	0.440 (0.053)	0.685 (0.040)	0.756 (0.054)	0.787; 0.007
	RotationForest	74.900 (3.300)	0.749 (0.033)	0.580 (0.052)	0.736 (0.034)	0.778 (0.058)	17.996; 0.277
	RandomForest	72.500 (2.500)	0.725 (0.025)	0.526 (0.060)	0.705 (0.034)	0.749 (0.043)	0.443; 0.005
diabetes	J-model (IM1)	74.723 (6.747)	0.747 (0.068)	0.660 (0.111)	0.737 (0.075)	0.799 (0.063)	6.195; 0.016
	J-model (IM2)	74.730 (5.648)	0.747 (0.057)	0.661 (0.091)	0.739 (0.063)	0.795 (0.083)	-
	J-model (IM3)	74.210 (6.035)	0.742 (0.061)	0.673 (0.108)	0.735 (0.069)	0.801 (0.071)	-
	J-model (IM4)	73.434 (7.448)	0.734 (0.074)	0.663 (0.124)	0.727 (0.083)	0.796 (0.076)	-
	J-model (IM6)	73.441 (4.807)	0.734 (0.048)	0.628 (0.081)	0.721 (0.055)	0.778 (0.081)	7.157; 0.000
	AdaBoost	74.351 (4.490)	0.744 (0.045)	0.654 (0.083)	0.735 (0.050)	0.805 (0.058)	0.206; 0.001
	Bagging	74.481 (3.126)	0.745 (0.031)	0.661 (0.060)	0.738 (0.034)	0.822 (0.045)	0.844; 0.002
	Decorate	73.833 (5.944)	0.738 (0.059)	0.653 (0.079)	0.732 (0.061)	0.803 (0.054)	2.860; 0.002
	LogitBoost	74.086 (2.714)	0.741 (0.027)	0.649 (0.053)	0.734 (0.030)	0.813 (0.039)	0.272; 0.001
	RandomCommittee	73.973 (4.199)	0.740 (0.042)	0.652 (0.059)	0.733 (0.042)	0.785 (0.046)	0.875; 0.009
	RandomSubSpace	74.614 (4.811)	0.746 (0.048)	0.626 (0.068)	0.732 (0.050)	0.812 (0.040)	0.517; 0.004
	RotationForest	76.177 (5.178)	0.762 (0.052)	0.662 (0.062)	0.753 (0.052)	0.821 (0.045)	2.440; 0.060
	RandomForest	73.841 (4.259)	0.738 (0.043)	0.640 (0.063)	0.729 (0.044)	0.778 (0.039)	0.797; 0.003
heart-statlog	J-model (IM1)	80.714 (7.178)	0.807 (0.072)	0.793 (0.100)	0.800 (0.080)	0.854 (0.098)	2.919; 0.000
	J-model (IM2)	81.428 (7.284)	0.814 (0.073)	0.802 (0.090)	0.810 (0.076)	0.862 (0.083)	-
	J-model (IM3)	81.428 (8.571)	0.814 (0.086)	0.811 (0.094)	0.811 (0.087)	0.840 (0.098)	-
	J-model (IM4)	82.143 (12.877)	0.821 (0.129)	0.804 (0.127)	0.814 (0.133)	0.881 (0.062)	-
	J-model (IM6)	80.714 (11.974)	0.807 (0.120)	0.801 (0.124)	0.803 (0.121)	0.840 (0.103)	3.308; 0.000
	AdaBoost	80.000 (4.743)	0.800 (0.047)	0.795 (0.067)	0.796 (0.052)	0.886 (0.055)	0.081; 0.001
	Bagging	78.889 (8.772)	0.789 (0.088)	0.779 (0.088)	0.786 (0.089)	0.886 (0.050)	0.255; 0.001
	Decorate	75.185 (6.839)	0.752 (0.068)	0.746 (0.073)	0.748 (0.071)	0.843 (0.074)	1.594; 0.001
	LogitBoost	82.222 (7.182)	0.822 (0.072)	0.814 (0.086)	0.819 (0.077)	0.888 (0.054)	0.122; 0.001
	RandomCommittee	80.370 (7.417)	0.804 (0.074)	0.793 (0.080)	0.800 (0.079)	0.873 (0.064)	0.180; 0.001
	RandomSubSpace	82.963 (4.743)	0.830 (0.047)	0.814 (0.060)	0.825 (0.051)	0.908 (0.050)	0.195; 0.001
	RotationForest	84.074 (7.417)	0.841 (0.074)	0.838 (0.079)	0.838 (0.078)	0.897 (0.041)	1.413; 0.043
	RandomForest	78.148 (5.092)	0.781 (0.051)	0.772 (0.056)	0.779 (0.052)	0.861 (0.054)	0.162; 0.002
hepatitis	J-model (IM1)	83.750 (11.250)	0.838 (0.113)	0.515 (0.353)	0.803 (0.140)	0.818 (0.205)	2.041; 0.000
	J-model (IM2)	81.250 (10.078)	0.813 (0.101)	0.357 (0.280)	0.761 (0.126)	0.835 (0.151)	-
	J-model (IM3)	80.000 (12.748)	0.800 (0.127)	0.488 (0.313)	0.765 (0.147)	0.810 (0.243)	-
	J-model (IM4)	81.250 (12.809)	0.812 (0.128)	0.442 (0.329)	0.771 (0.148)	0.796 (0.214)	-
	J-model (IM6)	82.500 (10.000)	0.825 (0.100)	0.446 (0.279)	0.785 (0.119)	0.849 (0.151)	2.383; 0.000
	AdaBoost	82.542 (5.850)	0.825 (0.058)	0.618 (0.161)	0.818 (0.057)	0.878 (0.076)	0.057; 0.000
	Bagging	83.167 (5.320)	0.832 (0.053)	0.393 (0.183)	0.785 (0.076)	0.825 (0.120)	0.134; 0.000
	Decorate	84.500 (8.656)	0.845 (0.087)	0.682 (0.150)	0.844 (0.081)	0.847 (0.101)	0.968; 0.001
	LogitBoost	81.917 (6.186)	0.819 (0.062)	0.558 (0.182)	0.805 (0.071)	0.841 (0.098)	0.058; 0.000
	RandomCommittee	84.583 (6.308)	0.846 (0.063)	0.589 (0.163)	0.834 (0.059)	0.853 (0.093)	0.088; 0.001
	RandomSubSpace	80.667 (2.669)	0.807 (0.027)	0.293 (0.123)	0.743 (0.040)	0.804 (0.159)	0.112; 0.001
	RotationForest	81.917 (6.788)	0.819 (0.068)	0.600 (0.178)	0.814 (0.066)	0.838 (0.125)	0.970; 0.032
	RandomForest	82.583 (5.836)	0.826 (0.058)	0.441 (0.223)	0.788 (0.076)	0.827 (0.100)	0.081; 0.000
ionosphere	J-model (IM1)	93.333 (4.157)	0.933 (0.041)	0.886 (0.074)	0.931 (0.044)	0.987 (0.013)	9.080; 0.000
	J-model (IM2)	93.889 (6.310)	0.939 (0.063)	0.905 (0.081)	0.938 (0.064)	0.970 (0.035)	-
	J-model (IM3)	93.889 (2.992)	0.939 (0.030)	0.897 (0.051)	0.938 (0.031)	0.968 (0.046)	-
	J-model (IM4)	94.444 (2.484)	0.944 (0.025)	0.917 (0.037)	0.944 (0.025)	0.966 (0.049)	-
	J-model (IM6)	93.333 (5.984)	0.933 (0.060)	0.909 (0.084)	0.932 (0.061)	0.965 (0.051)	10.043; 0.000
	AdaBoost	90.897 (3.950)	0.909 (0.040)	0.847 (0.063)	0.906 (0.042)	0.953 (0.040)	0.508; 0.001
	Bagging	90.897 (4.152)	0.909 (0.042)	0.876 (0.067)	0.907 (0.043)	0.936 (0.052)	1.039; 0.001
	Decorate	90.603 (2.548)	0.906 (0.025)	0.878 (0.035)	0.905 (0.025)	0.947 (0.043)	8.462; 0.002
	LogitBoost	91.175 (4.491)	0.912 (0.045)	0.884 (0.061)	0.911 (0.045)	0.951 (0.041)	0.464; 0.000
	RandomCommittee	92.603 (3.625)	0.926 (0.036)	0.905 (0.044)	0.926 (0.036)	0.976 (0.017)	0.494; 0.000
	RandomSubSpace	92.881 (2.622)	0.929 (0.026)	0.887 (0.035)	0.927 (0.027)	0.969 (0.031)	0.719; 0.003
	RotationForest	94.603 (2.294)	0.946 (0.023)	0.924 (0.036)	0.946 (0.023)	0.977 (0.023)	4.837; 0.119
	RandomForest	92.889 (3.634)	0.929 (0.036)	0.914 (0.042)	0.929 (0.036)	0.952 (0.029)	0.440; 0.000
kr-vs-kp	J-model (IM1)	99.375 (0.484)	0.994 (0.005)	0.993 (0.005)	0.994 (0.005)	0.999 (0.002)	40.541; 0.000
	J-model (IM2)	99.250 (0.673)	0.993 (0.007)	0.992 (0.007)	0.993 (0.007)	0.999 (0.001)	-
	J-model (IM3)	99.438 (0.438)	0.995 (0.004)	0.994 (0.005)	0.995 (0.004)	0.998 (0.004)	-
	J-model (IM4)	99.437 (0.710)	0.994 (0.007)	0.994 (0.007)	0.994 (0.007)	0.998 (0.003)	-
	J-model (IM6)	99.437 (0.337)	0.994 (0.003)	0.994 (0.004)	0.994 (0.003)	1.000 (0.001)	42.797; 0.000
	AdaBoost	93.836 (1.341)	0.938 (0.013)	0.936 (0.013)	0.938 (0.013)	0.955 (0.009)	0.904; 0.003
	Bagging	99.124 (0.460)	0.991 (0.005)	0.991 (0.005)	0.991 (0.005)	0.999 (0.000)	4.050; 0.008
	Decorate	99.312 (0.590)	0.993 (0.006)	0.993 (0.006)	0.993 (0.006)	0.998 (0.002)	36.469; 0.012

	LogitBoost	93.805 (1.347)	0.938 (0.013)	0.935 (0.013)	0.938 (0.013)	0.976 (0.008)	0.939; 0.008
	RandomCommittee	98.936 (0.659)	0.989 (0.007)	0.989 (0.007)	0.989 (0.007)	0.998 (0.002)	1.255; 0.021
	RandomSubSpace	96.059 (2.007)	0.961 (0.020)	0.959 (0.021)	0.960 (0.020)	0.992 (0.005)	4.525; 0.024
	RotationForest	99.219 (0.864)	0.992 (0.009)	0.992 (0.009)	0.992 (0.009)	0.998 (0.002)	36.417; 0.962
	RandomForest	98.811 (0.501)	0.988 (0.005)	0.987 (0.005)	0.988 (0.005)	0.999 (0.002)	0.995; 0.014
labor	J-model (IM1)	86.667 (16.330)	0.867 (0.163)	0.783 (0.299)	0.827 (0.216)	0.800 (0.332)	0.994; 0.000
	J-model (IM2)	86.667 (16.330)	0.867 (0.163)	0.783 (0.299)	0.827 (0.216)	0.950 (0.150)	-
	J-model (IM3)	86.667 (16.330)	0.867 (0.163)	0.783 (0.299)	0.827 (0.216)	0.850 (0.320)	-
	J-model (IM4)	90.000 (15.275)	0.900 (0.153)	0.850 (0.263)	0.873 (0.197)	0.900 (0.300)	-
	J-model (IM6)	86.667 (22.111)	0.867 (0.221)	0.833 (0.269)	0.823 (0.286)	0.850 (0.320)	1.072; 0.000
	AdaBoost	87.333 (16.180)	0.873 (0.162)	0.843 (0.225)	0.858 (0.183)	0.913 (0.142)	0.017; 0.000
	Bagging	86.333 (16.428)	0.863 (0.164)	0.812 (0.216)	0.840 (0.193)	0.919 (0.168)	0.041; 0.000
	Decorate	88.000 (10.873)	0.880 (0.109)	0.862 (0.139)	0.875 (0.111)	0.950 (0.083)	0.280; 0.001
	LogitBoost	89.667 (13.536)	0.897 (0.135)	0.853 (0.214)	0.880 (0.163)	0.988 (0.037)	0.023; 0.001
	RandomCommittee	89.667 (11.299)	0.897 (0.113)	0.853 (0.154)	0.889 (0.118)	0.975 (0.075)	0.035; 0.001
	RandomSubSpace	79.333 (21.333)	0.793 (0.213)	0.757 (0.264)	0.762 (0.244)	0.892 (0.146)	0.045; 0.000
	RotationForest	89.667 (13.536)	0.897 (0.135)	0.853 (0.214)	0.880 (0.163)	0.931 (0.144)	0.452; 0.138
	RandomForest	88.000 (13.182)	0.880 (0.132)	0.820 (0.215)	0.861 (0.159)	0.908 (0.187)	0.033; 0.000
mushroom	J-model (IM1)	100.000 (0.000)	1.000 (0.000)	1.000 (0.000)	1.000 (0.000)	1.000 (0.000)	121.649; 0.000
	J-model (IM2)	100.000 (0.000)	1.000 (0.000)	1.000 (0.000)	1.000 (0.000)	1.000 (0.000)	-
	J-model (IM3)	100.000 (0.000)	1.000 (0.000)	1.000 (0.000)	1.000 (0.000)	1.000 (0.000)	-
	J-model (IM4)	100.000 (0.000)	1.000 (0.000)	1.000 (0.000)	1.000 (0.000)	1.000 (0.000)	-
	J-model (IM6)	99.975 (0.074)	1.000 (0.001)	1.000 (0.001)	1.000 (0.001)	1.000 (0.000)	120.344; 0.000
	AdaBoost	96.197 (0.564)	0.962 (0.006)	0.963 (0.006)	0.962 (0.006)	0.995 (0.001)	1.861; 0.011
	Bagging	100.000 (0.000)	1.000 (0.000)	1.000 (0.000)	1.000 (0.000)	1.000 (0.000)	3.683; 0.009
	Decorate	100.000 (0.000)	1.000 (0.000)	1.000 (0.000)	1.000 (0.000)	1.000 (0.000)	32.022; 0.036
	LogitBoost	98.227 (0.427)	0.982 (0.004)	0.983 (0.004)	0.982 (0.004)	0.998 (0.001)	1.806; 0.011
	RandomCommittee	100.000 (0.000)	1.000 (0.000)	1.000 (0.000)	1.000 (0.000)	1.000 (0.000)	0.754; 0.016
	RandomSubSpace	100.000 (0.000)	1.000 (0.000)	1.000 (0.000)	1.000 (0.000)	1.000 (0.000)	2.927; 0.051
	RotationForest	100.000 (0.000)	1.000 (0.000)	1.000 (0.000)	1.000 (0.000)	1.000 (0.000)	142.115; 3.062
	RandomForest	100.000 (0.000)	1.000 (0.000)	1.000 (0.000)	1.000 (0.000)	1.000 (0.000)	0.782; 0.013
sonar	J-model (IM1)	84.364 (12.141)	0.843 (0.122)	0.850 (0.124)	0.842 (0.123)	0.899 (0.097)	9.346; 0.015
	J-model (IM2)	81.364 (15.626)	0.814 (0.156)	0.816 (0.156)	0.811 (0.160)	0.912 (0.100)	-
	J-model (IM3)	77.636 (12.755)	0.776 (0.128)	0.777 (0.137)	0.776 (0.128)	0.885 (0.096)	-
	J-model (IM4)	78.727 (13.967)	0.787 (0.140)	0.786 (0.141)	0.784 (0.144)	0.898 (0.097)	-
	J-model (IM6)	79.545 (10.002)	0.795 (0.100)	0.794 (0.100)	0.795 (0.100)	0.887 (0.095)	9.261; 0.000
	AdaBoost	71.667 (10.315)	0.717 (0.103)	0.713 (0.100)	0.708 (0.110)	0.861 (0.075)	0.492; 0.000
	Bagging	77.833 (10.164)	0.778 (0.102)	0.769 (0.103)	0.774 (0.103)	0.884 (0.063)	1.134; 0.000
	Decorate	78.333 (7.957)	0.783 (0.080)	0.781 (0.084)	0.781 (0.080)	0.902 (0.045)	7.366; 0.002
	LogitBoost	79.286 (9.417)	0.793 (0.094)	0.792 (0.094)	0.790 (0.097)	0.887 (0.070)	0.507; 0.001
	RandomCommittee	84.095 (7.211)	0.841 (0.072)	0.839 (0.074)	0.839 (0.074)	0.929 (0.040)	0.288; 0.000
	RandomSubSpace	77.952 (7.893)	0.780 (0.079)	0.773 (0.073)	0.776 (0.081)	0.868 (0.074)	0.667; 0.000
	RotationForest	80.810 (7.917)	0.808 (0.079)	0.800 (0.080)	0.804 (0.082)	0.903 (0.071)	4.713; 0.228
	RandomForest	80.738 (7.156)	0.807 (0.072)	0.815 (0.075)	0.807 (0.072)	0.911 (0.054)	0.248; 0.000
vote	J-model (IM1)	95.909 (3.776)	0.959 (0.038)	0.959 (0.042)	0.959 (0.038)	0.984 (0.029)	2.579; 0.000
	J-model (IM2)	95.455 (4.545)	0.955 (0.045)	0.956 (0.046)	0.955 (0.045)	0.990 (0.017)	-
	J-model (IM3)	95.455 (3.521)	0.955 (0.035)	0.951 (0.041)	0.955 (0.035)	0.985 (0.028)	-
	J-model (IM4)	95.000 (4.288)	0.950 (0.043)	0.943 (0.053)	0.950 (0.043)	0.982 (0.030)	-
	J-model (IM6)	95.455 (4.545)	0.955 (0.045)	0.956 (0.046)	0.955 (0.045)	0.989 (0.018)	2.529; 0.000
	AdaBoost	95.407 (3.233)	0.954 (0.032)	0.953 (0.033)	0.954 (0.032)	0.991 (0.009)	0.055; 0.002
	Bagging	95.872 (3.349)	0.959 (0.033)	0.956 (0.034)	0.959 (0.033)	0.984 (0.020)	0.187; 0.001
	Decorate	94.704 (2.731)	0.947 (0.027)	0.947 (0.026)	0.947 (0.027)	0.988 (0.015)	1.204; 0.001
	LogitBoost	95.412 (3.058)	0.954 (0.031)	0.956 (0.034)	0.954 (0.030)	0.992 (0.008)	0.073; 0.003
	RandomCommittee	96.321 (2.345)	0.963 (0.023)	0.957 (0.027)	0.963 (0.023)	0.988 (0.013)	0.147; 0.002
	RandomSubSpace	95.867 (2.667)	0.959 (0.027)	0.961 (0.029)	0.959 (0.027)	0.989 (0.008)	0.158; 0.001
	RotationForest	96.094 (3.091)	0.961 (0.031)	0.956 (0.034)	0.961 (0.031)	0.991 (0.007)	1.459; 0.071
	RandomForest	95.867 (2.863)	0.959 (0.029)	0.950 (0.035)	0.959 (0.029)	0.988 (0.014)	0.126; 0.002

We only measure time cost for J-model (IM1), J-model(IM6) and the other ensemble methods as we focus on the decreased test time cost of J-model (IM6).

7.6.3.1 Interaction model 1

Table 7.5: Standing of J-model (IM1) out of 9 ensemble classifiers on 13 data sets

Data set	Performance metrics				
	Accuracy	Sensitivity	Specificity	F-measure	AUC
breast-cancer	4th	4th	4th	5th	3rd ⋆
breast-w	2nd ⋆	2nd ⋆	2nd ⋆	2nd ⋆	1st ⋆
credit-a	1st ⋆	1st ⋆	1st ⋆	1st ⋆	3rd ⋆
credit-g	5th	5th	5th	5th	3rd ⋆
diabetes	2nd ⋆	2nd ⋆	3rd ⋆	3rd ⋆	7th
heart-statlog	4th	4th	5th	4th	8th
hepatitis	3rd ⋆	3rd ⋆	6th	6th	8th
ionosphere	2nd ⋆	2nd ⋆	5th	2nd ⋆	1st ⋆
kr-vs-kp	1st ⋆	1st ⋆	1st ⋆	1st ⋆	1st ⋆
labor	7th	7th	8th	8th	9th
mushroom	1st ⋆	1st ⋆	1st ⋆	1st ⋆	1st ⋆
sonar	1st ⋆	1st ⋆	1st ⋆	1st ⋆	5th
vote	3rd ⋆	3rd ⋆	2nd ⋆	3rd ⋆	8th

Table 7.5 summarise the standing of J-model when comparing with the 8 traditional ensemble methods. The stars besides the rankings are marked only for the 1st, the 2nd or the 3rd standing.

J-model got 9 starts out of 13 data sets for *accuracy*. When we notice the 1st and the 2nd standings, J-model got 7 starts. J-model got 9 stars for sensitivity (7 starts of the 1st and the 2nd); 7 (6) for specificity; 8 (6) for F-measure and 7 (4) for AUC.

7.6.3.2 Interaction model 2

The prediction performance is generally equivalent with or worse than J-model(IM1). We, however, noticed that J-model(IM2) shows better performance than J-model(IM1) for AUC.

7.6.3.3 Interaction model 3

J-model(IM3) showed that it is not better than J-model(IM1) at all. J-model(IM3) turned out to be an ineffective strategy for classification.

7.6.3.4 Interaction model 4 for higher specificity

IM4 is a specially designed interaction model for specificity. J-model(IM4) is better on credit-a, credit-g, diabetes, heart-statlog, ionosphere, labor (6 data sets); equivalent on breast-w, kr-vs-

kp, mushroom (3 data sets); worse on breast-cancer, hepatitis, sonar, vote (4 data sets) when comparing with J-model(IM1) for specificity.

Table 7.6: Standing of J-model (IM4) out of 9 ensemble classifiers for specificity

Data set	Specificity
breast-cancer	7th
breast-w	2nd ⋆
credit-a	1st ⋆
credit-g	3rd ⋆
diabetes	1st ⋆
heart-statlog	4th
hepatitis	6th
ionosphere	2nd ⋆
kr-vs-kp	1st ⋆
labor	5th
mushroom	1st ⋆
sonar	5th
vote	9th

Table 7.6 shows the standing of J-model(IM4) compared with the other 8 traditional ensemble methods. Stars are also marked for the 1st, the 2nd or the 3rd standing.

7.6.3.5 Interaction model 6 for lower test time cost

Interaction model 6 was designed to reduce the test time cost. In Table 7.4, the test time costs of J-model(IM1) and J-model(IM6) are compared. In J-model(IM6), the test time was dramatically reduced than J-model(IM1) although J-model(IM6) somewhat sacrificed its performance on the other performance metrics. The observation error can be ignored if it is less than 0.015 seconds because the JRuby implementation on Intel® Core™2 Duo CPU 2.67 GHz could not measure the time difference within 0.015 seconds. When comparing with the other ensemble methods, J-model(IM6) could not provide lower test time costs. This is because the test time costs of the other ensemble methods are already very small.

7.6.4 Conclusion

We applied several interaction models on J-model to benchmark data sets and compared their classification performance with other traditional and representative ensemble methods.

The results of IM1 showed that J-model(IM1) has better prediction performance than the others even though J-model uses non boosted classifiers from the pool. The traditional ensemble methods boost their performance through special mechanisms to get better accuracy and more diversity among the base classifiers.

IM2 showed better performance for a specific metric, AUC. IM2 was not designed for AUC intentionally, but it worked well for AUC. IM3 also was not designed for a specific metric. IM3, however, did not work well for any of the metrics. IM4 was designed intentionally for the specificity metric. It showed better results for specificity than IM1 gave and than the traditional ensemble methods did. IM6 is for reducing the time cost. It also worked for its purpose.

7.7 Realistic problem - virtual screening

7.7.1 Introduction

The experiments so far are on benchmark problems. We also applied J-model to solve a realistic problem. The virtual screening problem that we chose as a realistic problem has many more instances and attributes than the benchmark data sets. Moreover the virtual screening data set has highly imbalanced classes.

Virtual screening of bioassay data

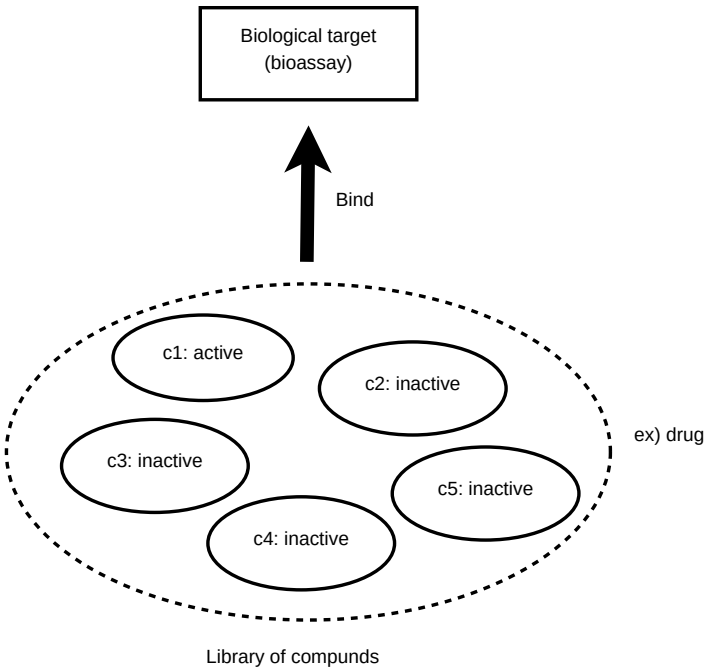

Figure 7.30: Selection of compounds against a biological target in virtual screening

Virtual screening [98] is the computational screening of chemical compounds. It complements the high-throughput screening (HTS) process and is used to aid the selection of compounds.

Highly-imbalanced data

The major challenge that we contact when we use machine learning techniques for bioassay virtual screening is that the data is highly-imbalanced. The data has a low ratio of *active* compounds to *inactive* compounds. The ratio is 1 active compound to 1000 inactive compounds on average [10]. Standard techniques are not very effective at building predictive models in this situation. We aim to find a robust and versatile classifier for imbalanced bioassay data.

7.7.2 Experimental setup

7.7.2.1 The PubChem bioassay data sets

Table 7.7: PubChem bioassay data sets

Index	Assay	Screening type	Compounds	Attributes	Active:inactive	Minority (active) class %
1	AID362	Primary	4279	144	60:4219	1.40
2	AID604	Primary	59788	154	212:59576	0.35
3	AID456	Primary	9982	153	27:9955	0.27
4	AID688	Primary	27198	154	248:26941	0.91
5	AID373	Primary	59788	154	62:59726	0.10
6	AID746	Primary	59788	154	366:59422	0.61
7	AID687	Primary	33067	153	94:32973	0.28
8	AID746&AID1284	Primary and confirmatory	59784	154	57:59727	0.10
9	AID604&AID644	Primary and confirmatory	59782	154	67:59715	0.11
10	AID373&AID439	Primary and confirmatory	59795	154	13:59782	0.02
11	AID687&AID721	Primary and confirmatory	33046	153	21:33046	0.06
12	AID1608	Confirmatory	1033	154	68:965	6.58
13	AID644	Confirmatory	206	100	67:139	32.52
14	AID1284	Confirmatory	362	103	57:305	15.75
15	AID439	Confirmatory	69	81	13:56	18.84
16	AID721	Confirmatory	94	87	21:73	22.34

The PubChem bioassay data sets [7] are highly-imbalanced bioassay data from different types of screening using high-throughput screening (HTS) technology. The data sets are 16 small to medium size ones. They have varying sizes and active classes. Table 7.7 shows a summary of the data sets used for study.

7.7.2.2 J-model settings

7.7.2.2.1 Pool preparation

7.7.2.2.1.1 Base classifier

- For non cost-sensitive version - RandomForest using the setting value in Table 7.1.
- For cost-sensitive version - RandomForest using the setting value in Table 7.1. We set cost matrices for it using the misclassification costs for false negatives in Table 7.8.

Table 7.8: Misclassification costs for false negatives of J-model

Assay	J-model
AID362	3000
AID604	50000
AID456	100000
AID688	8000
AID373	600000
AID746	18000
AID687	60000
AID746&AID1284	600000
AID604&AID644	900000
AID373&AID439	None
AID687&AID721	500000
AID1608	150
AID644	None
AID1284	8
AID439	None
AID721	None

7.7.2.2.1.2 Misclassification costs We set the misclassification cost of the cost-sensitive J-model when its FPR reaches about 20%. 20% FPR is an appropriate stop of permission. We did not set any cost for the assays indicated by *None*. On those assays, J-model shows results of under 20% FPR without cost setting.

7.7.2.2.1.3 Pool size We set the pool size of 64.

7.7.2.2.2 Ensemble size We set the ensemble size of 10

7.7.2.2.3 Choice of IM The bioassay classification should handle the problem of highly-imbalanced ratio of active and inactive classes. We used IM5 of Definition 6.10 to achieve higher TPR and lower FPR.

7.7.2.2.4 Number of interactions

- For non cost-sensitive version - We set 10000 interactions. This very high number was because active classes have very much smaller numbers than inactive ones. Enough interactions gives an opportunity to an ensemble to converge more toward the active classes. The more the ensemble takes active classes repeatedly, the more it becomes sensitive to the active classes.

- For cost-sensitive version - 200 interactions. Because the pool is composed with cost-sensitive RandomForests, we can set a much smaller number of interactions.

7.7.2.3 The other algorithms

7.7.2.3.1 Algorithms We re-used cost-sensitive Naive Bayes, cost-sensitive Random Forest, cost-sensitive SVM and cost-sensitive C4.5 whose results already have published in [86] for comparison with J-model. We also showed results of normal Random Forest.

Table 7.9: Misclassification costs for false negatives of the other CSC classifiers

Assay	Naive Bayes	Random Forest	SMO	J48
AID362	40	3000	150	285
AID604	40	Out of memory	250	650
AID456	18	100000	200	1000
AID688	34	Out of memory	78	220
AID373	20	Out of memory	2000	3000
AID746	25	Out of memory	100	450
AID687	50	Out of memory	250	680
AID746&AID1284	100	Out of memory	1000	1900
AID604&AID644	70	Out of memory	750	1500
AID373&AID439	70	Out of memory	9000	9500
AID687&AID721	700	Out of memory	6702	1900
AID1608	2	75	5	25
AID644	None	None	None	None
AID1284	None	8	2.7	2
AID439	None	None	None	None
AID721	None	None	None	None

7.7.2.3.2 Misclassification costs Classifiers give large variability following what misclassification costs are set. Table 7.9[2] shows the setting values of misclassification costs for the *false negatives* in order to achieve the maximum number of *true positives* with a false positive rate of fewer than 20% for each classifier. Random Forest classifiers require larger memory than the other classifiers. It utilises the bagging technique. In our results table, *out of memory* indicates that Random Forest could not be used for the experiments.

[2]From [86]

7.7.2.4 Evaluation technique

7.7.2.4.1 Split into train and test examples We split a assay data set to the train examples and test examples. The ratio is 80% (train) and 20% (test) of the total examples.

7.7.2.4.2 Performance metrics

- True Positives (TP) - In the bioassay case, active compounds correctly classified as active.
- False Positives (FP) - Inactive compounds incorrectly classified as active.
- False Negatives (FN) - Active compounds incorrectly classified as inactive.
- True Negatives (TN) - Inactive compounds correctly classified as inactive.
- True Positive Rate (TPR) - $TPR = TP/P = TP/(TP+FN)$. P is the positive (active) classes. The higher TPR value is preferred.
- False Positive Rate (FPR) - $FPR = FP/N = FP/(FP+TN)$. N is the negative (inactive) classes. The lower FPR value is preferred.
- *Accuracy* (ACC) - *Accuracy* is not a major performance metric for this sort of classification. We, however, include the results of *accuracy* for reference.

7.7.3 Results

Table 7.10: Virtual screening results

Assay	Algorithm	Performance metrics						
		TP	FN	FP	TN	TPR (%)	FPR (%)	Accuracy (%)
AID362	CSC Naive Bayes	9	3	161	683	75.00	19.08	80.84
	CSC Random Forest	10	2	159	685	83.33	18.84	81.19
	CSC SMO	9	3	126	718	75.00	14.93	84.93
	MetaCost J48	9	3	124	720	75.00	14.69	85.16
	Random Forest	1	11	1	843	8.33	0.12	98.60
	J-model (Non-CS,IM5,10000)	3	3	0	422	50.00	0.00	99.30
	J-model (CS,IM5,200) *	5	1	82	340	83.33	19.43	80.61
AID604	CSC Naive Bayes	23	19	2202	9713	54.76	18.48	81.43
	CSC Random Forest	-	-	-	-	-	-	-
	CSC SMO	27	15	2453	9462	64.29	20.59	79.36
	MetaCost J48	21	21	2401	9514	50.00	20.15	79.74
	Random Forest	4	38	3	11912	9.52	0.03	99.66
	J-model (Non-CS,IM5,10000)	0	21	0	5958	0.00	0.00	99.65
	J-model (CS,IM5,200) *	15	6	1214	4744	71.43	20.38	79.60
AID456	CSC Naive Bayes	3	2	296	1695	60.00	14.87	85.07
	CSC Random Forest	2	3	370	1621	40.00	18.58	81.31
	CSC SMO	3	2	133	1858	60.00	6.68	93.24
	MetaCost J48	2	3	312	1679	40.00	15.67	84.22
	Random Forest	0	5	0	1991	0.00	0.00	99.75
	J-model (Non-CS,IM5,10000)	0	2	0	996	0.00	0.00	99.80
	J-model (CS,IM5,200) *	2	0	217	779	100.00	21.79	78.26

AID688	CSC Naive Bayes	15	35	1048	4340	30.00	19.45	80.08
	CSC Random Forest	-	-	-	-	-	-	-
	CSC SMO	12	38	1094	4294	24.00	20.30	79.18
	MetaCost J48	8	42	1104	4284	16.00	20.49	78.93
	Random Forest	0	50	0	5388	0.00	0.00	99.08
	J-model (Non-CS,IM5,10000)	0	25	2	2692	0.00	0.07	99.01
	J-model (CS,IM5,200)	6	19	551	2143	24.00	20.45	79.04
AID373	CSC Naive Bayes	9	3	2146	9799	75.00	17.97	82.03
	CSC Random Forest	-	-	-	-	-	-	-
	CSC SMO	9	3	1966	9979	75.00	16.46	83.53
	MetaCost J48	9	3	1732	10213	75.00	14.50	85.49
	Random Forest	0	12	0	11945	0.00	0.00	99.90
	J-model (Non-CS,IM5,10000)	1	5	1	5972	16.67	0.02	99.90
	J-model (CS,IM5,200) *	5	1	1200	4773	83.33	20.09	79.91
AID746	CSC Naive Bayes	31	42	2462	9422	42.47	20.72	79.06
	CSC Random Forest	-	-	-	-	-	-	-
	CSC SMO	39	34	2085	9799	53.42	17.54	82.28
	MetaCost J48	46	27	2412	9472	63.01	20.30	79.60
	Random Forest	11	62	6	11878	15.07	0.05	99.43
	J-model (Non-CS,IM5,10000)	5	32	1	5941	13.51	0.02	99.45
	J-model (CS,IM5,200) *	32	5	1221	4721	86.49	20.55	79.49
AID687	CSC Naive Bayes	8	10	1251	5344	44.44	18.97	80.93
	CSC Random Forest	-	-	-	-	-	-	-
	CSC SMO	6	12	1213	5382	33.33	18.39	81.48
	MetaCost J48	5	13	1298	5297	27.78	19.68	80.18
	Random Forest	0	18	1	6594	0.00	0.02	99.71
	J-model (Non-CS,IM5,10000)	0	9	0	3298	0.00	0.00	99.73
	J-model (CS,IM5,200) *	4	5	585	2713	44.44	17.74	82.16
AID746&AID1284	CSC Naive Bayes	31	42	2462	9422	42.47	20.72	79.06
	CSC Random Forest	-	-	-	-	-	-	-
	CSC SMO	39	34	2085	9799	53.42	17.54	82.28
	MetaCost J48	46	27	2412	9472	63.01	20.30	79.60
	Random Forest	1	10	1	11944	9.09	0.01	99.91
	J-model (Non-CS,IM5,10000)	1	5	1	5971	16.67	0.02	99.90
	J-model (CS,IM5,200) *	4	2	1031	4941	66.67	17.26	82.72
AID604&AID644	CSC Naive Bayes	6	7	1542	10401	46.15	12.91	87.04
	CSC Random Forest	-	-	-	-	-	-	-
	CSC SMO	10	3	1422	10521	76.92	11.91	88.08
	MetaCost J48	7	6	1453	10490	53.85	12.17	87.80
	Random Forest	1	12	0	11943	7.69	0.00	99.90
	J-model (Non-CS,IM5,10000)	0	6	0	5972	0.00	0.00	99.90
	J-model (CS,IM5,200) *	4	2	1302	4670	66.67	21.80	78.19
AID373&AID439	CSC Naive Bayes	1	1	279	11678	50.00	2.33	97.66
	CSC Random Forest	-	-	-	-	-	-	-
	CSC SMO	1	1	1059	10898	50.00	8.86	91.14
	MetaCost J48	2	0	2111	9846	100.00	17.65	82.35
	Random Forest	1	1	0	11957	50.00	0.00	99.99
	J-model (Non-CS,IM5,10000)	0	1	0	5979	0.00	0.00	99.98
	J-model (CS,IM5,200)	0	1	0	5979	0.00	0.00	99.98
AID687&AID721	CSC Naive Bayes	2	2	959	5650	50.00	14.51	85.47
	CSC Random Forest	-	-	-	-	-	-	-
	CSC SMO	2	2	1484	5125	50.00	22.45	77.53
	MetaCost J48	2	2	625	5984	50.00	9.46	90.52
	Random Forest	0	4	0	6609	0.00	0.00	99.94
	J-model (Non-CS,IM5,10000)	0	2	0	3305	0.00	0.00	99.94
	J-model (CS,IM5,200) *	1	1	218	3087	50.00	6.60	93.38
AID1608	CSC Naive Bayes	3	10	37	156	23.08	19.17	77.18
	CSC Random Forest	4	9	16	177	30.77	8.29	87.86
	CSC SMO	4	9	17	176	30.77	8.81	87.38
	MetaCost J48	2	11	39	154	15.38	20.21	75.73
	Random Forest	0	13	1	192	0.00	0.52	93.20
	J-model (Non-CS,IM5,10000)	0	6	0	97	0.00	0.00	94.17
	J-model (CS,IM5,200)	1	5	17	80	16.67	17.53	78.64
AID644	CSC Naive Bayes	5	8	11	17	38.46	39.29	53.66
	CSC Random Forest	3	10	2	26	23.08	7.14	70.73
	CSC SMO	3	10	5	23	23.08	17.86	63.41
	MetaCost J48	5	8	8	20	38.46	28.57	60.98
	Random Forest	3	10	2	26	23.08	7.14	70.73
	J-model (Non-CS,IM5,10000)	3	4	1	13	42.86	7.14	76.19
	J-model (CS*,IM5,200) *	3	4	0	14	42.86	0.00	80.95

AID1284	CSC Naive Bayes	3	8	16	45	27.27	26.23	66.67
	CSC Random Forest	5	6	11	50	45.45	18.03	76.39
	CSC SMO	4	7	8	53	36.36	13.11	79.17
	MetaCost J48	6	5	8	53	54.55	13.11	81.94
	Random Forest	3	8	2	59	27.27	3.28	86.11
	J-model (Non-CS,IM5,10000)	3	3	2	28	50.00	6.67	86.11
	J-model (CS,IM5,200) *	4	2	5	25	66.67	16.67	80.56
AID439	CSC Naive Bayes	2	0	3	8	100.00	27.27	76.92
	CSC Random Forest	1	1	2	9	50.00	18.18	76.92
	CSC SMO	1	1	1	10	50.00	9.09	84.62
	MetaCost J48	1	1	2	9	50.00	18.18	76.92
	Random Forest	1	1	2	9	50.00	18.18	76.92
	J-model (Non-CS,IM5,10000)	1	0	1	5	100.00	16.67	85.71
	J-model (CS*,IM5,200) *	1	0	1	5	100.00	16.67	85.71
AID721	CSC Naive Bayes	0	4	4	10	0.00	28.57	55.56
	CSC Random Forest	0	4	3	11	0.00	21.43	61.11
	CSC SMO	0	4	2	12	0.00	14.29	66.67
	MetaCost J48	0	4	2	12	0.00	14.29	66.67
	Random Forest	0	4	3	11	0.00	21.43	61.11
	J-model (Non-CS,IM5,10000)	0	2	2	5	0.00	28.57	55.56
	J-model (CS*,IM5,200)	0	2	2	5	0.00	28.57	55.56

The values of TP, FN, FP and TN on J-models can be considered to be half that of the other algorithms. The reason is that J-model splits the original test set into the query set and J-model's test set.

7.7.3.1 Results of Random Forest

Random Forest showed extremely low performance on the bioassay data sets. It nearly cannot give the correct answers about true positives. Random Forest is useless as a classifier for the bioassay problem. This non cost-sensitive Random Forest (an original Random Forest labelled Random Forest in Table) can be viewed as a baseline for performance compared with the results of non-CS and CS J-models. This is because both of non-CS and CS versions of J-model filled their classifier pools with the Random Forest template classifiers.

7.7.3.2 Results of non-CS J-model

We applied J-model of the interaction model 5. The pool was composed with classifiers trained without cost sensitivity. We set the number of interactions as 10000 to give J-model great opportunity of exploration for true positives.

The performance became higher than the performance of Random Forest. But the performance was much lower than the other cost-sensitive algorithms of CSC Naive Bayes, CSC Random Forest, CSC SMO and MetaCost J48.

7.7.3.3 Results of CS J-model

We set just 200 interactions for J-model of IM5 because we filled in the pool with cost-sensitive classifiers. J-model got the 1st standing for three fourth of the assays (12 times out of 16 assays) among 5 cost-sensitive algorithms. J-model got 2nd for AID688 and AID604&AID644; 4th for AID1608; the last for AID373&AID439.

7.7.4 Conclusion

The issue that is how to deal with highly imbalanced data is a major challenge in machine learning research [19, 104]. The issue arises in many real-world domains where the target examples are rare in the data.

Cost-sensitive J-model showed very good performance for the imbalanced data. The pool members which J-model used were just cost-sensitive classifiers. They are not boosted classifiers by the traditional ensemble algorithms.

Cost-sensitive J-model interacted just 200 times. With a small number of interactions, J-model could achieve good performance.

As we can see in the case of cost-sensitive Random Forest, cost-sensitive Random Forest could not give results because it ran out of memory. J-model did not suffer from this space complexity problem as J-model does not need extra memory space for the ensemble.

Table 7.1: Base classifier templates for pool generation

Classifier template	Options of classifier template [setting value]	Filter options of classifier template [setting value]	Base classifier of the template	Options of base classifier [setting value]
AdaBoostM1	· Percentage of weight mass to base training on [100]		DecisionStump	
Bagging	· Size of each bag, as a percentage of the training set size [100]		REPTree	· Set minimum number of instances per leaf [2] · Set minimum numeric class variance proportion of train variance for split [0.0010] · Number of folds for reduced error pruning [3] · Maximum tree depth [1]
Decorate	· Desired size of ensemble [1] · Factor that determines number of artificial examples to generate. Specified proportional to training set size [1.0]		J48	· Set confidence threshold for pruning [0.25] · Set minimum number of instances per leaf [2]
LogitBoost	· Percentage of weight mass to base training on [100] · Number of folds for internal cross-validation [0] · Number of runs for internal cross-validation [1] · Threshold on the improvement of the likelihood [1.7976931348623157E308] · Shrinkage parameter [1.0]		DecisionStump	
RandomCommittee			RandomTree	· Number of attributes to randomly investigate [0] · Set minimum number of instances per leaf [1.0]
RandomForest	· Number of features to consider [0]			
RandomSubSpace	· Size of each subspace [0.5]		REPTree	Same with REPTree for Bagging
RotationForest	· Minimum size of a group of attributes [3] · Maximum size of a group of attributes [3] · Percentage of instances to be removed [50]	Filter specification [PrincipalComponents] · Retain enough PC attributes to account for this proportion of variance in the original data [1.0] · Maximum number of attributes to include in transformed attribute names [5] · Maximum number of PC attributes to retain [1]	J48	Same with REPTree for Decorate

Chapter 8

Discussion

8.1 Balance between exploration and exploitation

We defined the rank calculation for query interaction in Algorithm 5.1. The peer ranking service recommends supporting peers based on this rank calculation for query interactions. The definition that we set is the following.

$$R_Q(p) = C(p, \ominus) \tag{8.1}$$

This definition (8.1) was designed to promote exploration among peers and exploitation of higher scored peers. It is sensitive to the number of minuses of each peer.

$$R_Q(p) = 1 - (C(p, \oplus) - C(p, \ominus)) \tag{8.2}$$

We can try to apply another definition of rank calculation for query interaction instead of (8.1). The definition (8.2) considers both the number of pluses and minuses of each peer to calculate rank. It looks reasonable because it reflects both sides of scores (plus and minus). When a peer gets more pluses for queries, the peer has the higher rank.

There is, however, a serious defect when this definition (8.2) is applied to the ranking process. Figure 8.1, 8.2 and 8.3 show the results of the number of peer being selected over interactions when we apply (8.2).

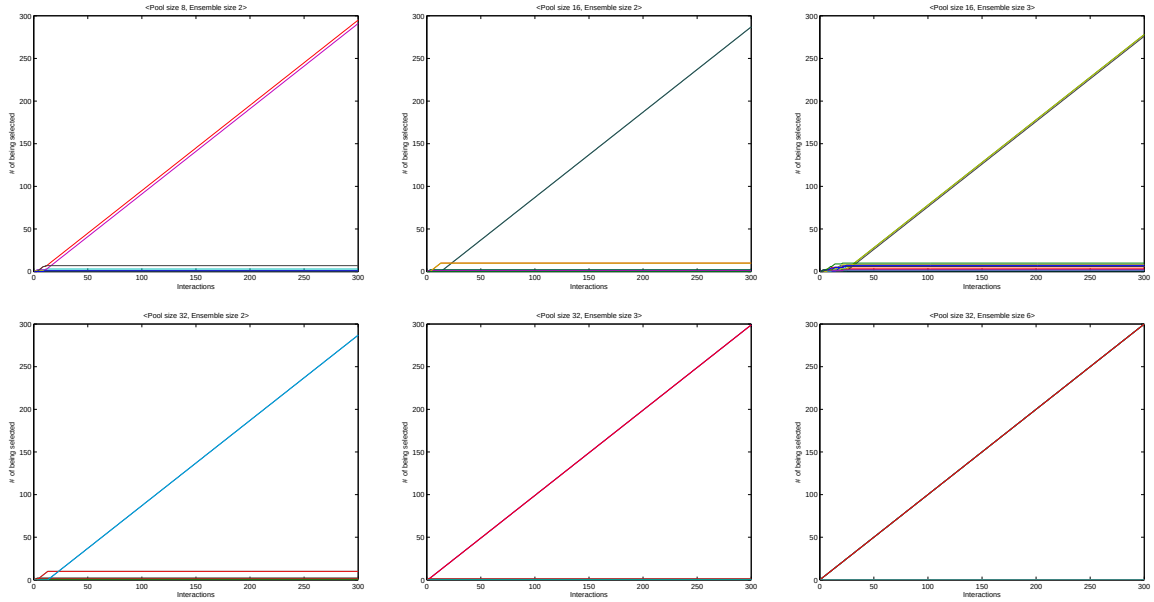

Figure 8.1: Number of peers being selected over interactions with breast-cancer using rank calculation (8.2)

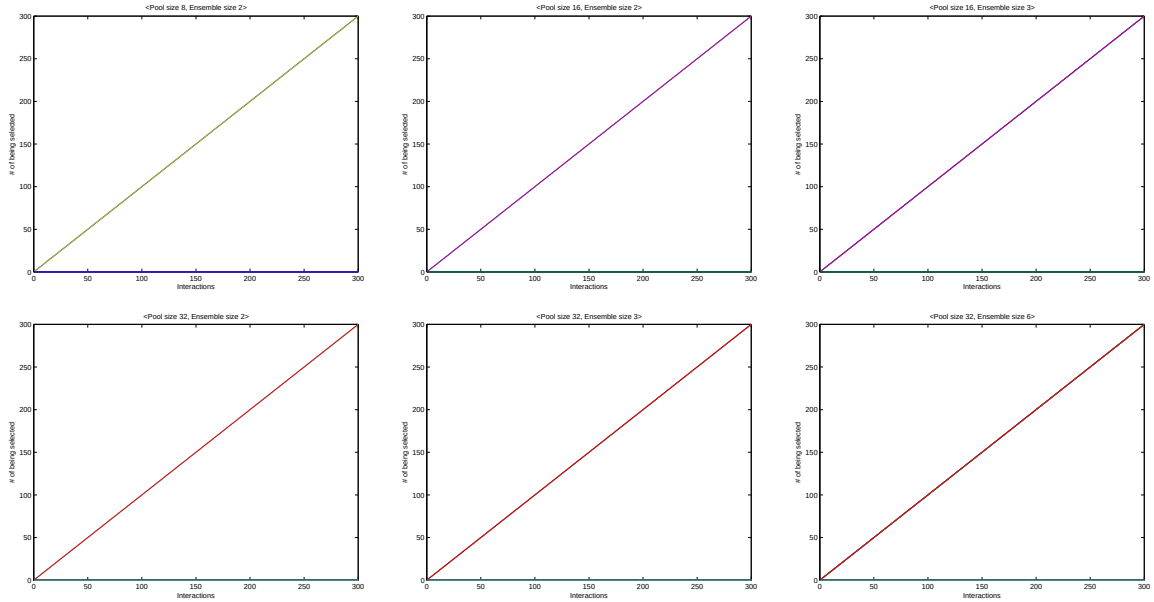

Figure 8.2: Number of peers being selected over interactions with kr-vs-kp using rank calculation (8.2)

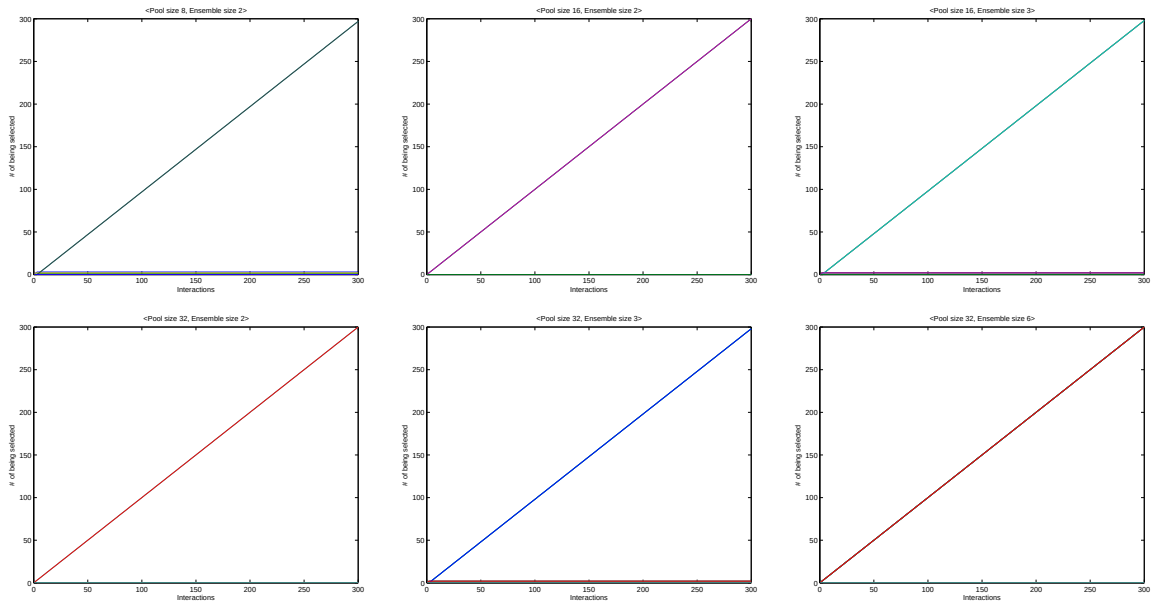

Figure 8.3: Number of peers being selected over interactions with labor using rank calculation (8.2)

In each figure above, always, the same peers are repeatedly selected over interactions except in very early steps. The other peers cannot be selected at all. There is no exploration among peers, only exploitation for the specific peers.

The reason why the other peers cannot be selected is that the scores of the specific peers (higher scored peers) are never under the scores of the lower scored peers. For example, peer 1 (having 70% accuracy) and peer 2 (having 70% accuracy) on average get 7 pluses and 3 minuses for queries for the first 10 interactions. They continually get $+4$ on 10 interactions, $+8$ on the next 10 interactions and so on. They always are selected by the rank calculation because they always have the highest scores and repeated selections reinforce this. The lower scored peers have no chance to be selected. The following graphs of 8.4, 8.5 and 8.6 show this problem more apparently.

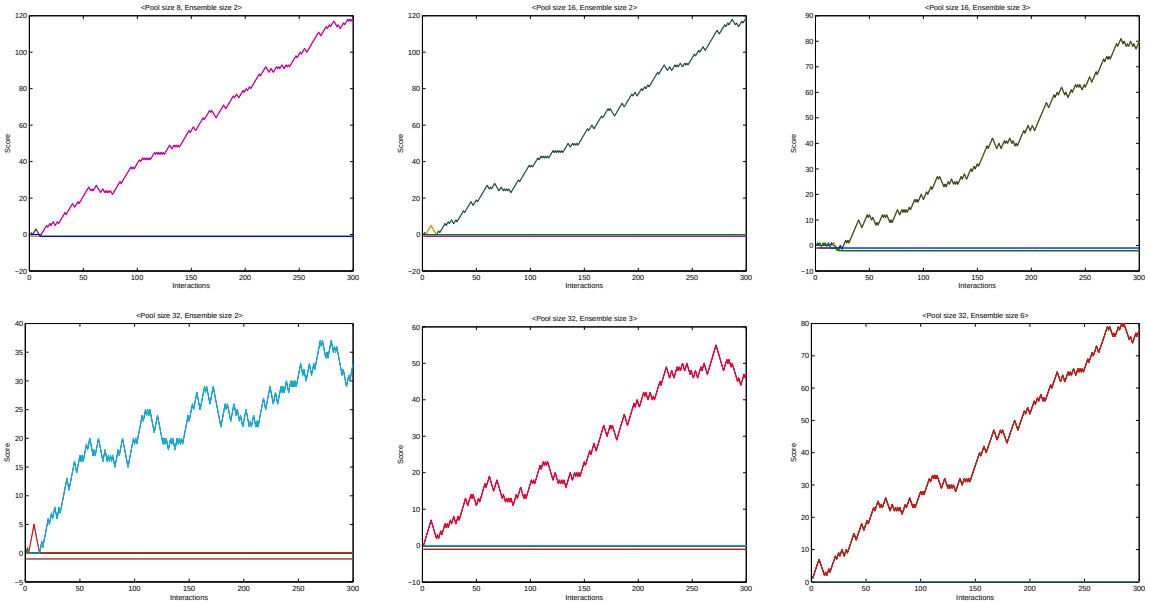

Figure 8.4: Score over interactions with breast-cancer using rank calculation (8.2)

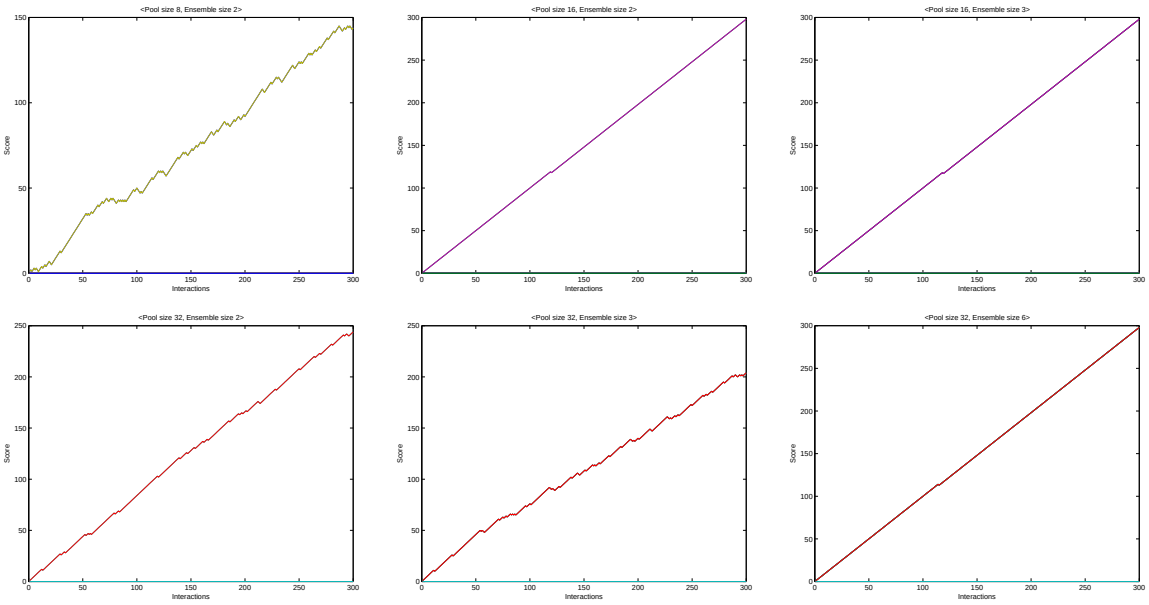

Figure 8.5: Score over interactions with kr-vs-kp using rank calculation (8.2)

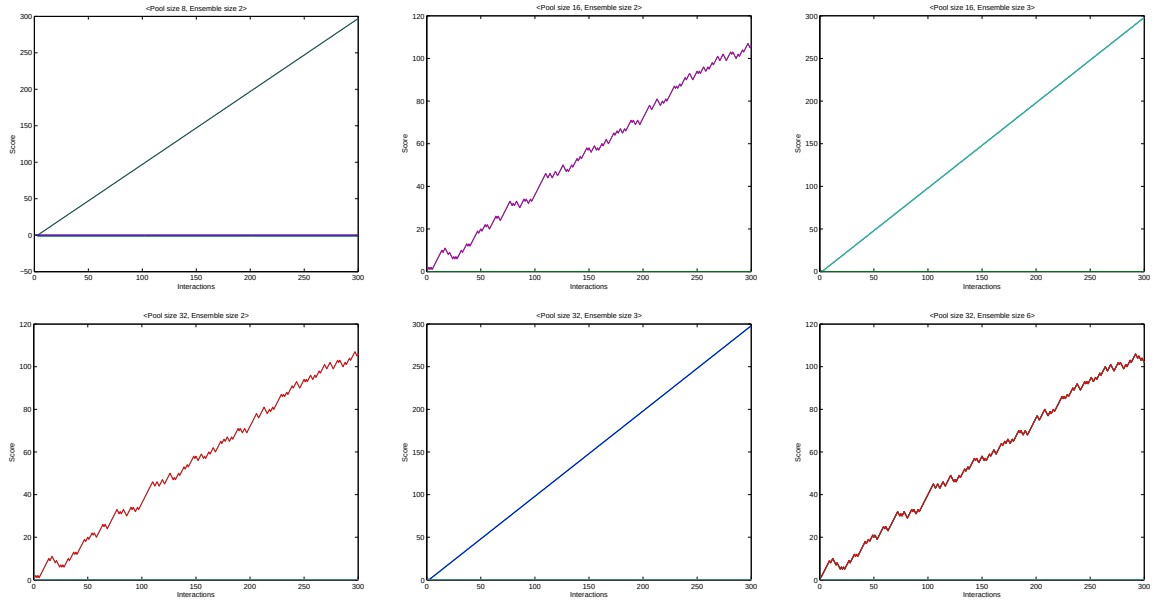

Figure 8.6: Score over interactions with labor using rank calculation (8.2)

Therefore we need to refine rank calculation to allow more exploration. That definition must address several defects that a simple ranking for more exploration might have. One of the defects is that a definition always explores all peers but it does not maintain exploitation for higher scored peers. Another defect is that a new definition may need more time to finish a peer separation than a reasonable time. Last, we might have less confidence about higher scored peers because the ability of the peers has not been repeatedly verified with enough queries. The most extreme definition of this sort of a simple exploration is visiting all of peers randomly.

In the next section, we discuss better techniques of exploring peers and exploiting higher scored peers as realised in our initial definition of rank calculation of (8.1).

8.2 More exploration on less accurate ensembles and more exploitation on more accurate ensembles

8.2.1 More frequent moving on less accurate ensembles

According to the rank calculation for query interaction in Algorithm 5.1, the peer ranking algorithm recommends supporting peers. Current supporting peers might be changed if the previous supporting peers predicted a wrong answer.

Let us assume that the potential accuracy value of a current ensemble is 0.3. This ensemble classifier is a less accurate ensemble classifier. Based on the rank calculation for query interaction, the chance that the ensemble is selected again on the next interaction round is small because the ensemble has less possibility to give a correct prediction for a current query example.

This means that the ensemble selection might move from a less accurate ensemble to another ensemble rapidly. This feature makes J-model explore more less accurate ensembles.

8.2.2 More opportunities to be confident on more accurate ensembles

Let us assume that the potential accuracy value of a current ensemble is 0.7. This ensemble classifier is a more accurate ensemble classifier. Based on the rank calculation for query interaction, the chance that the ensemble is selected again on the next interaction round is large because the ensemble has more possibility to give a correct prediction for a current query example.

This means that the ensemble selection might stay with the current ensemble for the next interaction round. A more accurate ensemble still has greater probability over interactions. It means that J-model has more opportunities to be confident of the performance of the ensemble because the ensemble is frequently confirmed with queries.

8.2.3 Under dynamic condition

We experimented the peer separation under dynamic conditions in Section 7.3.4. The separation process in the figures looked more noisy than the process under static conditions.

The dynamic condition we set in Section 7.1.4 makes 25% randomly selected classifiers from the pool miss in every interaction. When this is applied to higher scored peers, they become to be confirmed with queries less frequently (so there is less exploitation). When this condition is applied to lower scored peers, the selection which would have been on them moves to other peers more rapidly (so there is more exploration). Those two tendencies made the separation under dynamic conditions more noisy.

8.3 Big falls in learning curves

On the learning curve experiment of kr-vs-kp (Figure 7.28), we can see that some of the learning curves have big falls. The big falls only occur in early interactions.

We might guess that the reason is that the test examples are too small. *Accuracy* is calculated as how many corrections there are out of the total tests. The small size makes big gaps among the values for *accuracy*. However, that reasoning is not applicable on the case of kr-vs-kp. First, kr-vs-kp has enough instances (3196 instances). Second, the big falls appear only in early interactions and the gaps of falls decrease after that or are being removed.

We suggest more likely reasons. First, in the early interaction phase, historical verification on current higher scored peers with queries is not firmly established. So a discontinuous move from the current group of higher scored peers to a new group of different higher scored peers might result in a sudden and big fall of accuracy in a learning curve. Second, especially in kr-vs-kp, the big falls look dramatic because the average accuracy of prediction in kr-vs-kp is very high (it is over 97%). So this makes the big falls look extreme on small changes of performance.

8.4 Cyclic curves in learning curves

We could see several cyclic curves in the labor experiment (Figure 7.29). The reason that they occur can be explained as follows.

The size of the query data set for labor is very small. Total instances are just 57. J-model adapts to this very small number of queries for an ensemble with the same queries frequently being used for validating an ensemble. If the ensemble *A* gives an wrong answer, another ensemble *B* is recommended by the peer ranking algorithm. If *B* gives an wrong answer, recommendation then moves to *A* cyclically. This is definitely a bad thing under static conditions (so we should avoid this sort of circulating) but it might sometimes be useful under dynamic conditions (eg. if the learners might independently improve performance between steps in the cycle).

8.5 Accuracy of the pool in J-model

J-model separates optimal peers for better classification from a peer pool. A peer pool is composed of diverse classification services. We showed that J-model gives better performance in the benchmark experiments of the section 7.6. Its performance is from non-boosted (not intentionally tuned to training examples) member classifiers.

However, in the experiment of a realistic classification problem in Section 7.7, the pool that was composed of only non-tuned peers (non-CS J-model of Section 7.7.3.2) could not give a good performance even though we set the enough number of interactions as high as 10000.

We could confirm that diversity in a pool is essential for performance but minimal quality among classifiers in the pool is also needed for boosting performance. So we needed cost-sensitive classification services as pool members and could get a good performance from them.

8.6 Appropriate ensemble size

We used 20% of a pool size as an ensemble size for the benchmark and the realistic classification problems. Our choice of ensemble size is based on the Pareto principle. The principle states that roughly 80% of the effects come from 20% of the causes for many events. This effect sometimes is identified in multi-agent systems, for example in SugarScape [29] which simulated wealth distribution.

However, it is interesting to consider what happens if we vary the ratio of ensemble to pool size. Let us see what happens if the ensemble size is too small or too big. When the size is small, the search space will be large. This means that J-model is likely to visit many candidate ensembles in a pool. So its search space in this respect is larger. Peer separation and its convergence to an optima in J-model are based on the history of getting scores of peers. In a big search space, we will need more interactions to get enough scoring history.

If the ensemble size is big, the search space will be small. This means that the number of candidate ensembles is small. So J-model needs less time to search but the size of ensembles obscures differences between peers. In this case, the scoring history on peers may not be distributed properly to discriminate the peers.

8.7 Minimal parameterisation

Learning parameters influence the performance and effectiveness of machine learning. These parameters make the individual machine learning systems adapt to the particulars of a training set. Tuning parameters, however, is an expensive and complex task.

The parameters that J-model has are the ensemble size and the number of interactions. A pool is given as an environment and an interaction model defines the system of agent coordination upon which J-model works. The ensemble size is determined by the Pareto principle. So we have one parameter that needs to be tuned, the number of interactions.

We set the number of interactions as 200 when we experimented with J-model for the benchmark and realistic classification problems in Chapter 7. The value was determined based on the results of Section 7.3. Higher scored peers were separated from other peers and they normally kept their dominance over the other peers after at least 200 interactions had been done. This, however, is a heuristic approach to determining the proper number of interactions. It would be useful to have an automatic method for predicting the number of interactions needed for stability.

We suspect that a general and effective basis for this will be to measure change of the gap between average scores of the higher scored peers and the lower scored peers as shown in Section 7.3.3.3 and 7.3.4.3, since this gives us an indirect measure of their performance. If we use performance as a measure for the termination condition, the number of interactions is automatically determined when J-model gets to an expected performance.

8.8 Conclusion

In this chapter, we have discussed issues being categorised to three groups. First, we suggested discussions about what features on a reputation mechanism should be required in our architecture in Section 8.1 and 8.2. Second, we explained why big falls and cyclic curves appear in learning curves and the meaning of those in Section 8.3 and 8.4. Last, we discussed for how J-model can give better prediction results on the quality of classifier pool, appropriate ensemble sizes and setting the number of interactions in Section 8.5, 8.6 and 8.7.

Chapter 9

Related work

9.1 Distributed ensemble classification

Distributed ensemble classification is a research sub-domain for solving classification problems in distributed data mining. Distributed ensemble classification is implemented by applying traditional ensemble methods straightforwardly to distributed environments or building smarter versions of ensemble methods for distributed environments.

9.1.1 Distributed data mining

The development of information and communication technologies has brought us a large number of different and distributed computing devices and data sources. The Internet, geographically distributed information systems such as the earth observing system of NASA[1], *sensor networks*, *grids* are examples of such distributed environments.

When we apply a traditional (centralised) knowledge discovery process to distributed environments, it requires us to gather all the distributed sources within a central repository for central processing. This is neither effective nor feasible for several reasons: storage cost, communication cost, computational cost, and private and sensitive data issues. Distributed data mining (DDM) includes algorithms, methods and systems that efficiently discover knowledge in distributed environments.

In DDM, discovering knowledge takes place in each local or distributed site and then global

[1] http://eos.gsfc.nasa.gov/

knowledge is integrated from the local knowledge at a global level. There are two different approaches to synchronising global knowledge among local sites. One approach is that a global site sends global knowledge back to local sites, so that they are updated with the global knowledge. The other approach is that local knowledge is broadcast to all other local sites, so that they share global knowledge which we hope will converge over time.

9.1.2 Distributed classification

Approaches for distributed classification are mostly inspired from ensemble methods such as Stacking [103], Voting [50, 54] and Boosting. Some approaches to apply ensemble methods to distributed environments are straightforward. The other approaches use smarter methods to reduce communication and coordination costs.

Chan and Stolfo [18] applied Stacking ensemble method to DDM through adopting their meta-learning technique. Their meta-learning technique is to construct a meta-level training data set through combining distributed training examples. Their methodology showed better performance for a number of domains. Knowledge Probing [40] uses an independent data set called the probing set on the meta-learning technique of Chan and Stolfo. The probing set is used to select an appropriate ensemble model for a problem.

Several techniques have been suggested for building a single classifier from local classifiers which have been trained on an individual distributed set. Hall, Chawla and Bowyer [42, 43] suggested a technique of assembling a decision tree with distributed sub decision trees represented as rule sets. Each sub decision trees learns disjoint data. The rule combination continually takes the union of the distributed rule sets resolving any conflicts.

Fan, Stolfo and Zhang [32] introduced d-sampling AdaBoost which is an extended version of AdaBoost for DDM. At each boosting round, an individual weak learner in a distributed site trains its local data set. Then a distribution of weights D_t is calculated from the results of the current round and D_t is applied to all the distributed data sets. Experiments showed that their DDM version of AdaBoost gives comparable or better performance than a single machine learning algorithm trained with the union of distributed data in most cases. However it gave comparable performance to a single classical boosting algorithm only in limited cases. Lazarevic and Obradovic [55] presented a distributed boosting algorithm in which weak learners of distributed sites learns in parallel at each round. Weak learners share their local D_ts by broadcasting the local values one another. Their experiments showed that the algorithm brings

comparable or slightly better classification performance than a single boosting algorithm with the union of distributed data sets.

9.2 Agent-based distributed data mining

Agent-based distributed data mining research started from the motivation for bringing benefits such as abilities to solve autonomy and scalability problems that the agent technology can give to distributed data mining.

9.2.1 Introduction

Distributed data mining research has taught us that cooperation among distributed data mining processes may give effective mining results without using centralised data mining approaches. This naturally led us to adopting the agent technology for the development of cooperative DDM called agent-based DDM. Agent-based DDM provides the inherent feature of agents of being autonomous and adaptive. These features are intended to solve autonomy and scalability problems of DDM. Agents perform various mining operations instead of humans and computing devices that are operated by humans and collaborate with other agents. Agent-based DDM systems aim to cope with data mining tasks in distributed, heterogeneous and massive data environments.

9.2.2 Benefits from agents for DDM

The following items are benefits that data mining agents (DM agents) give for DDM.

- Autonomy of data source
  A DM agent is a modular process in a data management system. A DM agent accesses data sources and gathers knowledge from the data sources under given constraints with autonomy.
- Scalable DDM
  For massive distributed data, a DDM system can let DM agents take each distributed data set and perform data mining in their local sites. Then their mined data are merged in the original DDM system.

- Multi-strategy DDM

  There are cases where we wish to obtain greater effectiveness for complex data mining tasks by combining DM agents using different strategies for individual complex tasks instead of applying a single strategy.

- Collaborative DDM

  There may be conflicting combinations among DM agents who learned their local data independently. Collaborative DM agents have ability to negotiate their own opinions with each other and may give a collaborated global opinion.

- Dynamicity in open distributed data environments

  Open distributed data environments in which the availability of data sites must be considered and their content may change at any time have issues of how to discover and select relevant data sources for performing DM tasks. DM agents can be used under these conditions. DM agents adaptively select data sources based on their selection criteria such as availability, quality, form and network load of data sources.

9.2.3 Learning strategy for agent-based DDM

Several systems have been suggested for agent-based data mining. These systems can be categorised according to their learning strategy into three types of central learning, meta learning and hybrid learning. Meta-learning and hybrid-learning DM systems are more appropriate for distributed data mining because central-learning systems gather data at a central site and build a single model.

9.2.3.1 Meta-learning strategy

Meta-learning methods have been used particularly for classification and regression tasks [96, 9]. For classification tasks, a meta-learning method has three main steps. In the first step, it generates classifiers at each site using machine learning algorithms for classification. Next, it gathers the generated classifiers at a central site. In the last step, it builds the final classifier (meta-classifier) through combining the gathered classifiers.

One of the most well known agent-based meta-learning approaches is the METAL project[2]. This project is for helping users to gain a ranking of suitability among DM algorithms through

[2]http://www.metal-kdd.org

an on-line advisory system. AgentDiscover, a multi-agent system for knowledge discovery and data mining (KDD) [75], was introduced. It uses task-based reasoning for problem solving.

The most mature agent-based meta-learning systems are JAM and BODHI. Both of the systems are intended for data classification.

JAM [89] is a Java-implemented multi-agent system designed based on meta-learning DDM. JAM agents learn heterogeneous databases using different machine learning algorithms such as Ripper, CART, ID3, C4.5, Bayes and WEPBLS. JAM agents may be resident in a single site or imported agents from other peer sites in the system. JAM offers a group of meta-learning agents which combines multiple classifier agents at different sites into a meta-classifier. In many cases. these meta-classifiers give improved predictive accuracy.

BODHI [48] is a framework for performing collective DM tasks on heterogeneous data such as supervised inductive distributed function learning and regression. BODHI guarantees to get a correct local and global data model with low network communication load. The framework provides message exchange and runtime environments for mobile agents running at each local site. The mining process is distributed to the local sites and agents move between the sites on demand. Each agent transports its state, data and knowledge. A central facilitator agent has a responsibility of initialising and coordinating the communication and control flow among the agents.

9.2.3.2 Hybrid-learning strategy

A hybrid-learning method combines local and remote learning for building a model [94]. Papyrus [5] is an example of hybrid-learning systems. Papyrus is a specialised DDM system for clusters of heterogeneous data sites and meta-clusters. It supports several sorts of predictive models including C4.5. In contrast to JAM and BODHI, Papyrus can not only move models from site to site, but can also move data when a suggested strategy requires. Each cluster has one primary node with which agents access and control clusters. The overall clustering task is coordinated in a central root site or across a distributed network of cluster access points in a peer-to-peer manner. Papyrus supports various model combination methods and a special markup language is used to describe the meta-description of data, models and intermediate results.

9.3 Collaborative multi-agent learning

Multi-agent learning is a technique to build complex multi-agent systems in a dynamic environment. Collaborative multi-agent learning is a special type of multi-agent learning. In collaborative multi-agent learning, agents work together as a group to improve their accuracy at a given learning task.

9.3.1 Multi-agent learning

Multi-agent learning has been defined as learning through the interaction between multiple intelligent agents [47]. This multi-agent learning was developed from our attempt to build complex multi-agent systems that operate in dynamic environments. It is extremely difficult to design complex multi-agent systems with robustness in advance. This difficulty naturally led us to develop multi-agent systems that adapt and learn through experience.

Multi-agent learning is different from standard machine learning. Standard machine learning methods work under assumption that a single learner or agent has all relevant knowledge locally. In multi-agent systems, this assumption is not available. Relevant knowledge such as training experience and background information is distributed among agents in a multi-agent systems. Also domain constraints such as privacy and cost may require a multi-agent approach.

9.3.2 Collaborative multi-agent learning

Collaborative multi-agent learning is a special type of multi-agent learning, in which agents work together as a group or team to improve their accuracy at a given learning task. Agents actively communicate or interact with one another during the learning process in order to be collaborative. The main issue in the interaction is how agents learn accurately without exposing their knowledge to a central agent.

Collaborative multi-agent leaning is basically different from ensemble learning such as bagging and boosting. Ensemble learning methods combine the predictions from N independent learners through voting schemes. Variance across learners helps to improve overall accuracy. This ensemble learning approach, however, will not work properly when a target problem cannot be learnt by individual member classifiers. Meanwhile, collaborative learning allows a group of learners to learn those sorts of target problems. In collaborative learning, a distributed

situation is assumed.

9.3.3 Research on collaborative multi-agent learning

Weiß and Dillenbourg [101] express an important opinion that the true potential for multi-agent learning is obtained through dynamic forms of interactivity. This opinion offers a general perspective across the multi-agent learning research domain.

Many existing works of collaborative learning were performed on collective versions of reinforcement learning. Weiß [100] and Tan [91] independently pointed that collaborative learning can be improved through information exchange between agent and Tan additionally suggested social adaptation of agents for the improvement. Whitehead and Ballard [102] provided a learning architecture based on mutual observation.

A large number of other works have suggested collaborative versions of Q learning since the works of Weiss, Tan and Whitehead. Clouse [22] showed that collaborative improvement can be achieved through letting agents ask for help with one another. Chalkiadakis and Boutilier [17] suggested a collaborative model to explore the space of solutions. Szer and Charpillet [90] defined an algorithm to broadcast intermediate learning results and investigated effects that the circulation of different quantities of information makes. Vu, Powers and Shoham [97] studied agent coordination. Their results explore the minimum levels of performance in each agent needed for their collaboration.

There is a notable work on collaborative multi-agent learning that is not based on reinforcement learning. Prasad [76] redesigned collaborative multi-agent learning as a parametric problem. A group of agents cooperatively searches a composite search space. To goal is to find globally optimal solutions. Agents share their local data with one another when conflicts arise. This information is reused during search rounds.

Some researchers studied collaborative multi-agent learning through linking with other machine learning techniques. Modi and Shen [66] suggested a distributed collaborative learning algorithm for classification in situations where some of the information is privately closed. Ontañón and Plaza [69] proposed cooperation techniques for case-based reasoning. Nunes and Oliveira [68] provided an advice exchange system where the circulation of information occurs among agents having different learning algorithms. Graça and Gaspar [39] gave the results of the performance of opportunistic non-learning agents that receive information from learning agents. They concluded that agents having different tasks and roles can improve global

performance.

9.4 Open multi-agent systems

Researches on open multi-agent systems are for providing multi-agent systems working with open, dynamic and heterogeneous agents.

9.4.1 Introduction

In recent years, open multi-agent systems (OMASs) have gained importance in the study of distributed AI. Agents participating in OMAS may proactively join and leave the system at any time and they may independently have been implemented by different designers.

A main problem in OMAS is how to coordinate the ability of these open, dynamic and heterogeneous agents. Additionally, the agent coordination may not possible to be designed at design time. This coordination has a intrinsic feature of being arranged at runtime. According to characteristics of systems, two different sorts of coordination approaches are applicable respectively. First, methods to prescribe and enforce behaviour of each agent are applicable if a system has an explicit global goal to achieve and there exists an authority enabling to enforce the prescribed behaviour. Second, societal structures can be applied if a system does not have any global goal or an authority. In such systems, agents interact with one another and a more efficient behaviour can be obtained through their coordination of interactions for a global goal. This brings the difficult task of how to decide which agents an agent interacts with.

9.4.2 Research on open multi-agent systems

Most work has used prescriptive structures in order to regulate OMAS. Artikis [4] introduced an infrastructure for dynamic protocol specifications where a specification may change at runtime by agents participating in an OMAS.

Notable works based on structural adaptation have been suggested. Kota, Gibbins and Jennings [52] provided a decentralised approach for structural adaptation. Their method realised an implicit adaptation of agents for their structural relationship by which task allocation processes improve.

There exist many works by which a MAS changes its organisation during execution. Deloach, Oyenan and Matson [25] provided a framework in which a MAS organisation re-organise at runtime. Dignum, Dignum and Sonenberg [26] and Wang, Liang and Zhao [99] also presented re-organisation of organisation structures.

Hübner, Vercounter and Boissier [46] suggested a collective process reputation for trust management by coordination artifacts publishing and providing objective evaluations that agent calculate.

9.5 Service choreography workflows

We now provide an overview of workflow technology for coordinating distributed services. We focus on a choreography approach because it is more adaptive and scalable for changing and uncertain services.

9.5.1 Introduction

Workflow technology is one of the major approaches for coordinating distributed services as a group. In service-oriented architectures, services are loosely coupled and independent from one another and accordingly they offer a greater degree of flexibility and scalability for evolving applications. Coordination of services is appropriate when a shared goal can be achieved through collaboration of the services.

9.5.2 Service orchestration and service choreography

There are two main architectural approaches in which workflows are executed; service orchestration and service choreography. This criterion specifies whether workflow is executed in a centralised (orchestration) or a distributed (choreography) manner.

In service orchestration, a single process (the process acts as a controller) executes the activities and the other passive processes (services) are called by the single process. Service orchestration workflows are defined through orchestration languages such as WS-BPEL, YAWL and XPDL.

In service choreography, the activities are executed by active participating services that communicate or interact via messages with one another. Emergent collaboration among them naturally

arises. Service choreography workflows are described through choreography languages such as WS-CDL, WSCI and OWL-S.

9.5.3 Choreography languages

WS-CDL, WSCI and OWL-S are XML-based languages and they support WSDL[3] [21, 20] which is the founded standard to describe Web services.

9.5.3.1 WS-CDL

The Web Service Choreography Description Language (WS-CDL) describes peer-to-peer collaborations of Web services. This description defines the common behaviour of participating services and the ordered message interchanges. In WS-CDL, the collaboration between Web services arises by the ordering and constraint rules with which services agree. The elements in WS-CDL are as follows.

- Role - A role enumerates a potential behaviour of a participant within an interaction.
- Channel - A channel specifies where and how information between participants is exchanged.
- Relationship - A relationship identifies the mutual obligations that have to be implemented to succeed.

9.5.3.2 WSCI

The Web Service Choreography Interface (WSCI) describes the interface of a Web service in a choreographed interaction. This interface declares the flow of messages exchanged by the Web service. One WSCI interface defines the observable behaviour of one Web service. Temporal and logical dependencies in the flow of messages represent this behaviour. A WSCI choreography consists of a set of interfaces.

[3] http://www.w3.org/TR/wsdl

9.5.3.3 OWL-S

The Ontology Web Language for Services (OWL-S) was developed to support the concept of Semantic Web from the DARPA Agent Markup Language (DAML[4]). The purpose of this ontology language is to automate the discovery, invocation, composition, interoperation and monitoring of Web services. The ontology proposed by OWL-S is designed to provide three essential sorts of information about services.

- Service profile - what the service provides (for being discovered)
- Service process model - how the service is used (for being used)
- Service grounding - how to access the service (for being used)

9.6 Ensemble selection

Ensemble selection aims to reduce ensemble sizes prior to classifier combination. Ensemble selection decreases computational overhead from a large number of member classifiers and may acquire better predictive performance from classifiers having various predictive performance levels.

9.6.1 Introduction

Ensemble methods typically have two phases for learning: the generation of multiple classifiers and their combination. Ensemble selection is an additional intermediate phase to reduce the ensemble size prior to combination. Ensemble selection gives us two benefits: efficiency and predictive performance. Managing a large number of member classifiers in an ensemble brings computational overhead such as large memory requirements and computational cost. Ensemble selection can reduce this computational overhead. Member classifiers might be composed of both high and low predictive performance models but low predictive performance models can badly affect the performance of an ensemble. Ensemble selection removes these low performing models.

[4]http://www.daml.org/

9.6.2 Ensemble selection algorithms

Ensemble selection methods that have been proposed so far can be categorised into four categories: search-based, clustering-based, ranking-based and other methods.

9.6.2.1 Search-based methods

Search-based methods are the most direct approach for ensemble selection. They heuristically select different classifier subsets in the classifier search space based on some metric and each candidate is evaluated. Search-based methods can be divided to greedy search and stochastic search based on the search paradigm.

Greedy search The greedy search paradigm is the most popular category of ensemble selection. Greedy search tries to find a globally best classifier subset by searching the classifier subset space.

In the research of Fan *et al.* [31], Martínez-Muñoz and Suárez [61], Caruana *et al.* [16] and the Reduce-Error Pruning with Backfitting method [62], forward selection was used for searching the classifier subset space [59].

Backward elimination was used in the AID thinning and concurrency thinning algorithms [6].

Stochastic search Stochastic search gives a chance to select a random ensemble candidate for a next round. This helps ensemble selection process to avoid getting stuck in local optima.

The GASEN-b algorithm [107] applied a genetic algorithm (GA) to perform stochastic search in the space of classifier subsets. A bit string represents an ensemble. One bit indicates a classifier. Corresponding bits determine which classifiers become members of an ensemble. The operations of GP such as crossover and mutation are applied to ensembles. The performance of an ensemble is evaluated as the fitness value.

Partalas *et al.* [72] employed Q-learning for stochastic search. In Q-learning, selecting n classifiers for an ensemble is transformed into letting an agent learn an optimal policy of taking n actions of including or excluding classifiers.

9.6.2.2 Clustering-based methods

Clustering-based methods have two-step stages. The first step is to apply a clustering algorithm to discover sets of classifiers that look giving similar predictions. The second step is to prune each cluster separately.

Giacinto, Roli and Fumera [38] applied Hierarchical Agglomerative Clustering (HAC). It was required to define a distance metric between classifiers to use this clustering algorithm. They defined this metric as the probability of the coincident error level between classifiers and used a validation set to calculate the error levels. In pruning within a cluster, a single representative cluster is selected. The selected classifier has the maximal average distance from all other clusters. For making an ensemble from the selected classifiers pruned in the individual clusters, all the combinations of the selected classifiers are evaluated using a validation set based on majority voting as the combination method. The combination that has achieved the highest classification accuracy becomes the final ensemble.

Lazarevic and Obradovic [56] used the *k*-means algorithm to make clusters of classifiers. In this method, to determine the value of k (the number of clusters) is an issue. They continually increased the number of clusters until diversity among clusters began to decrease to solve the given problem. They then pruned classifiers of each cluster based on a pre-defined threshold of classification accuracy.

Fu, Hu and Zhao [37] also used the *k*-means algorithm for clustering classifiers. They pruned each cluster by selecting a single classifier that has the highest classification accuracy as Giacinto, Roli and Fumera did and determined the number of clusters as Lazarevic and Obradovic did.

9.6.2.3 Ranking-based methods

Ranking-based methods give an order to classifiers in an ensemble according to some evaluation metric and select classifiers based on the order.

In the Orientation Ordering algorithm [62], classifiers get their orders based on the angle between their signature vector and reference vector. The signature vector of a classifier c is a $|D|$-dimensional vector. Each element has the value $+1$ if $c(x_i) = y_i$ and -1 if $c(x_i) \neq y_i$. x is a validation example, y is an actual class value and i is an index of an example in a validation set. The reference vector is a vertical vector of an ensemble signature vector which is an average signature value of all classifiers in an ensemble. Classifiers whose angle is less than $\pi/2$

become the members of a final ensemble.

9.6.2.4 Other methods

Two sorts of approaches that do not belong to the three categories above have been introduced. The first approach [93, 92] is based on statistical procedures for directly selecting a subset of classifiers. The other approach [106] is based on semi-definite programming (SDP), more specifically quadratic integer programming.

9.7 Social reputation mechanisms

Social reputation mechanisms are used to improve the reliability and performance of electronic societies through rating reputation of the members.

9.7.1 Introduction

The research on computational reputation mechanisms is a discipline that has gained significant attention in recent years. Its aim is to increase the reliability and performance of introduced electronic communities.

There are two sorts of social evaluations; local (subjective) reputation and global reputation. In local reputation, reputation inferences are performed from the perspective of another agent and thus each agent in the network may have multiple reputation values. Local reputation is subjective by nature. Mechanisms such as ReGreT [85], RepAge [84], Sierra-Debenham model [88], AFRAS [15] and FIRE [27] are based on local reputation. In global reputation, the reputation of each agent is computed from the perspective of the whole network and thus each agent is associated to a single reputation value. An individual agent has a public reputation in the community. Examples that follow global reputation are online auctions such as eBay[5] and Amazon Auctions[6], laboratory models such as Sporas [105], and Web related algorithms such as PageRank [13], HITS [51] and TrustRank [41].

[5] http://www.eBay.com
[6] http://auctions.amazon.com

9.7.2 Mechanisms based on global reputation

9.7.2.1 Online reputation mechanisms

eBay and Amazon Auctions are representative examples of online marketplaces using reputation mechanisms. On eBay, the reputation mechanism is based on the ratings that users add after the completion of a transaction. The user can choose one of the three possible values: positive (1), negative (-1) or neutral (0). The reputation value is calculated as the sum of those ratings in the last six months. Amazon Auctions use a mean value as the reputation value.

9.7.2.2 Sporas

Sporas is an evolved version of the online reputation mechanisms. Sporas has two main features for handling reputation. First, only the most recent rating between two users is used for computing the reputation value. Second, users having a very high reputation are likely to maintain their ratings while users having a low reputation obtain big rating changes. Sporas measures the reliability of the users' reputation based on the standard deviation of reputation values. Sporas is more robust to changes of the user behaviour and the reliability measure helps the reputation value more usable.

9.7.2.3 Link-based algorithms

There are many link-based algorithms for finding authoritative, influential, central and reputable nodes on a network. These algorithms can generally be applied to any sort of network.

9.7.2.3.1 PageRank The PageRank algorithm is inspired from how the number of citations determine the relevance of a paper in the scientific community. PageRank conceptually maps a link from a Page A to a page B into a vote of the page A for the page B. The formula to calculate the PageRank value of the page is the following[7]:

$$PR(p_i) = \frac{1-d}{N} + d \sum_{p_j \in M(p_i)} \frac{PR(p_j)}{L(P_j)} \tag{9.1}$$

[7] From http://en.wikipedia.org/wiki/PageRank

p is a Web page. N is the total number of pages. $M(p_i)$ is the set of pages that link to p_i. $L(P_j)$ is the number of outbound links on page p_j. d is a damping factor.

9.7.2.3.2 HITS The HITS algorithm also considers the relevance of a Web page based on its links, like PageRank. However, this algorithm only use a subset of pages instead of using any page that links to the target page. *Authority* pages and *hub* pages are the selected pages to be used. The Web page authors provide an algorithm to determine an *authority* page that is linked from many good pages. Authors also define a *hub* page that has links to many authoritative pages.

Chapter 10

Conclusions

We live in an environment in which things are being generated in greater numbers and are connecting one another much more frequently, faster and broader than before. Things are autonomous, adaptive and communicative with sensing and processing. We can call these *smart objects*. Examples of smart objects are wireless sensors, ambient communication devices, household appliances and mobile medical devices.

We know that a higher degree of *smartness* can be derived from interoperation of those smart objects. The higher degree of smartness is obtained through shared knowledge.

Our J-model architecture provides a practical platform to share one form of valuable knowledge among smart objects that perform classification.

10.1 Hypothesis confirmation

We now return to our original hypotheses in the introduction chapter and check whether the research hypotheses of this thesis have been confirmed.

The hypotheses, which we introduced in Section 1.3, are

1. J-model's prediction performance approaches the performance of traditional ensemble methods.
2. J-model's prediction performance approaches the performance of traditional ensemble methods in practical time.
3. J-model is applicable to realistic learning problems.

4. Minimal parameterisation is required for J-model prediction.

Each hypothesis has been confirmed as follows.

1. In the experiments of the section 7.6, we compared J-model's prediction performance with other representative ensemble methods on the metrics of accuracy, sensitivity, specificity, F-measure and the area under curve (AUC) using standard machine learning benchmark data sets. The results showed that J-model's performance is comparable to the performances of the other traditional ensemble methods.

2. In the benchmark comparisons in Section 7.6, we commonly set 200 as the number of interactions. The number of interactions we set was determined based on the peer separation experiments of Section 7.3. After 200 interactions, peers were reliably separated from each other and their score orders became stable. 200 interactions finishes within one second of physical time in the benchmarks. This is a practical time for us to get classification results.

   Formally, J-model's time complexity for coordination is $O(N^2)$. As shown in a pseudo-code representing an ensemble coordination process of J-model in Section 4.3, J-model's coordination process has two loops of an outer and an inner ones. The outer loop is for the number of interactions and the inner loop is for the size of an ensemble (the number of roles defined in an interaction model is the same as the size of an ensemble as each member of an ensemble takes its corresponding role). This time complexity analysis supports that J-model's coordination process is competitive.

3. We applied J-model to virtual screening classification problem in Section 7.7. The results for true positive rates on these highly imbalanced data sets was remarkably successful. The results was obtained by using 200 interactions. J-model did not suffer from a memory space problem for these large size data sets. This tells us that J-model is applicable to realistic learning problems.

4. We discussed the parameterisation issue of J-model in Section 8.7 of the discussion chapter. J-model required only the number of interactions as a parameter and the value of the interaction parameter might be able to be determined automatically.

10.2 Contributions to knowledge

The key contributions of this thesis are as follows:

- First, we showed that service choreography coordination can be an effective ensemble learning process for classifiers in an open context. In experiments, its classification results are better than the results from traditional ensemble methods and they are given in practical time.
- Second, we gave more attention to task-oriented machine learning as distinct from previous passive learning techniques. Learning tasks are implemented by designing interaction models.
- Third, we designed a reputation mechanism showing network effects and a power-law distribution for machine learning. The mechanism recommends more appropriate classifiers from a classifier pool. Formally, we defined the peer ranking algorithm suitable for general machine learning classification. It is robust for classification services in an open context and decides reputation of services based on the result of interaction. The peer ranking mechanism is general and independent of the design of individual interaction models or classifiers.

10.3 Weaknesses on our work

Despite of contributions of our work, there remains weak points on our achievements as follows:

- We chose a heuristic selection approach for the number of interactions.

  As we discussed this problem in Section 8.7, we need an automatic method that adaptively determines the appropriate number of interactions at runtime. This issue is essential because the number of interactions determines the degree of convergence of an ensemble. If the convergence is premature, a less verified ensemble might be selected as a final ensemble classifier. Meanwhile, if the convergence is too mature with an excess number of interactions, a selected ensemble might be over-fitted to query examples.

- A single peer ranking service might not be appropriate for a huge classifier pool.

  We used a single peer ranking service to calculate ranks of peers. The single peer ranking service might encounter a problem if it should give ranks for a huge number of peers because requirements of the service for memory and time may be a bottleneck in our architecture.

This problem can be solved by using multiple peer ranking services. Each service takes one of the peer clusters. A global rank can be calculated by merging local ranks that individual peer ranking services calculated for their peer clusters.

10.4 Future work

10.4.1 Adaptive parameterisation

We discussed J-model's parameterisation issues in the discussion chapter (Section 8.7). In the section, we suggested that the number of interactions which is needed for enough peer separation can be determined based on the size of query examples and changes of the gap between average scores of the higher scored peer and the lower scored peers. J-model could be made more sensitive to what happens during its interactions, for example by adjusting the selection balance among peers. That is, we have a plan to make J-model itself adjust its parameters adaptively and automatically at run-time.

10.4.2 General peer ranking algorithm

We evaluated J-model with standard machine learning benchmark data sets in Section 7.6 and a realistic classification problem of virtual screening in Section 7.7. The results showed that J-model gives good prediction performance. We will evaluate J-model with other realistic problems. The peer ranking algorithm of J-model is generally applicable at least for the problems that we used in this thesis. We would like to measure how well the peer ranking algorithm works for further realistic problems. If the ranking algorithm does not show sufficient performance for a broader range of problems, we need to adjust the ranking algorithm (for example by adaptive parameterisation) so that the ranking algorithm gives balanced exploration and exploitation and network effects and a power-law distribution for peers across many classification problems.

10.4.3 Regression, clustering and reinforcement learning

We utilised J-model for classification problems. We expect that we can expand the application of J-model to other sorts of machine learning such as regression and clustering problems and

reinforcement learning. J-model is a general ensemble learning architecture. So we expect that J-model can be applied to such learning problems simply through preparing an appropriate classifier pool and specified interaction models without changing any architectural components of J-model or re-defining the peer ranking algorithm.

10.4.4 Mathematical analysis

We would like to deepen our understanding of J-model's learning process by the peer ranking algorithm through the development of a more extensive abstract study of its mathematical properties. We think that random process theory can be one of the most appropriate candidates for this purpose. Through analysing J-model learning mathematically, we can understand its learning process at a precise and fundamental level and the understanding may help us to be able to design better J-model learning.

Bibliography

[1] J. Abian, M. Atencia, P. Besana, L. Bernacchioni, D. Gerloff, S. Leung, J. Magasin, A.P. de Pinninck, X. Quan, D. Robertson, et al. Bioinformatics interaction models. *Deliverable D6.3, OpenKnowledge*, 2008.

[2] C. Anderson. *The long tail: Why the future of business is selling less of more*. Hyperion Books, 2008.

[3] A. Arkin, S. Askary, S. Fordin, W. Jekeli, K. Kawaguchi, D. Orchard, S. Pogliani, K. Riemer, S. Struble, P. Takacsi-Nagy, et al. Web service choreography interface (WSCI) 1.0. *Standards proposal by BEA Systems, Intalio, SAP, and Sun Microsystems*, 2002.

[4] A. Artikis. Dynamic protocols for open agent systems. In *Proceedings of The 8th International Conference on Autonomous Agents and Multiagent Systems-Volume 1*, pages 97–104. International Foundation for Autonomous Agents and Multiagent Systems, 2009.

[5] S. Bailey, R. Grossman, H. Sivakumar, and A. Turinsky. Papyrus: a system for data mining over local and wide area clusters and super-clusters. In *Proceedings of the 1999 ACM/IEEE conference on Supercomputing (CDROM)*, Supercomputing '99, New York, NY, USA, 1999. ACM.

[6] R.E. Banfield, L.O. Hall, K.W. Bowyer, and W.P. Kegelmeyer. Ensemble diversity measures and their application to thinning. *Information Fusion*, 6(1):49–62, 2005.

[7] E.E. Bolton, Y. Wang, P.A. Thiessen, and S.H. Bryant. Pubchem: integrated platform of small molecules and biological activities. *Annual Reports in Computational Chemistry*, 4:217–241, 2008.

[8] K.D. Borne. Astroinformatics: A 21st century approach to astronomy. *Arxiv preprint arXiv:0909.3892*, 2009.

[9] J. Botía, A. Gómez-Skarmeta, M. Valdés, and A. Padilla. Metala: A meta-learning architecture. *Computational Intelligence. Theory and Applications*, pages 688–698, 2001.

[10] D. Bradley. Dealing with a data dilemma. *Nature Reviews: Drug Discovery*, 7:632–633, 2008.

[11] Leo Breiman. Bagging predictors. *Machine Learning*, 24(2):123–140, 1996.

[12] Leo Breiman. Random forests. *Machine learning*, 45(1):5–32, 2001.

[13] S. Brin and L. Page. The anatomy of a large-scale hypertextual web search engine. *Computer networks and ISDN systems*, 30(1-7):107–117, 1998.

[14] A. Broder, R. Kumar, F. Maghoul, P. Raghavan, S. Rajagopalan, R. Stata, A. Tomkins, and J. Wiener. Graph structure in the web. *Computer networks*, 33(1):309–320, 2000.

[15] J. Carbo, JM Molina, and J. Davila. Trust management through fuzzy reputation. *International Journal of Cooperative Information Systems*, 12(1):135–155, 2003.

[16] R. Caruana, A. Niculescu-Mizil, G. Crew, and A. Ksikes. Ensemble selection from libraries of models. In *Proceedings of the twenty-first international conference on Machine learning*, page 18. ACM, 2004.

[17] G. Chalkiadakis and C. Boutilier. Coordination in multiagent reinforcement learning: a bayesian approach. In *Proceedings of the second international joint conference on Autonomous agents and multiagent systems*, pages 709–716. ACM, 2003.

[18] Philip Chan and Salvatore J. Stolfo. Toward parallel and distributed learning by meta-learning. In *In AAAI Workshop in Knowledge Discovery in Databases*, pages 227–240, 1993.

[19] N.V. Chawla, N. Japkowicz, and A. Kotcz. Editorial: special issue on learning from imbalanced data sets. *ACM SIGKDD Explorations Newsletter*, 6(1):1–6, 2004.

[20] R. Chinnici, J.J. Moreau, A. Ryman, and S. Weerawarana. Web services description language (WSDL) version 2.0 part 1: Core language. *W3C Recommendation*, 26, 2007.

[21] E. Christensen, F. Curbera, G. Meredith, S. Weerawarana, et al. Web services description language (WSDL) 1.1, 2001.

[22] J.A. Clouse. Learning from an automated training agent. In *Adaptation and Learning in Multiagent Systems*, 1996.

[23] C. Cortes and V. Vapnik. Support-vector networks. *Machine learning*, 20(3):273–297, 1995.

[24] J.A.N. de Caritat Marquis de Condorcet. *Essai sur l'application de l'analyse à la probabilité des decisions*. De l'imprimerie royale, 1785.

[25] S.A. Deloach, W.H. Oyenan, and E.T. Matson. A capabilities-based model for adaptive organizations. *Autonomous Agents and Multi-Agent Systems*, 16(1):13–56, 2008.

[26] Virginia Dignum, Frank Dignum, and Liz Sonenberg. Towards dynamic reorganization of agent societies. In *In Proceedings of Workshop on Coordination in Emergent Agent Societies*, pages 22–27, 2004.

[27] T. Dong-Huynha, N. Jennings, and N. Shadbolt. Fire: An integrated trust and reputation model for open multi-agent systems. In *ECAI 2004: 16th European Conference on Artificial Intelligence, August 22-27, 2004, Valencia, Spain: including Prestigious Applicants [sic] of Intelligent Systems (PAIS 2004): proceedings*, volume 110, page 18. Ios Pr Inc, 2004.

[28] D. Easley and J. Kleinberg. *Networks, crowds, and markets*. Cambridge Univ Press, 2010.

[29] J.M. Epstein and R. Axtell. *Growing artificial societies: social science from the bottom up*. The MIT Press, 1996.

[30] T. Erl. *Service-oriented architecture*. Prentice Hall, 2004.

[31] Wei Fan, Fang Chu, Haixun Wang, and Philip S. Yu. Pruning and dynamic scheduling of cost-sensitive ensembles. In *Eighteenth national conference on Artificial intelligence*, pages 146–151, Menlo Park, CA, USA, 2002. American Association for Artificial Intelligence.

[32] Wei Fan, S.J. Stolfo, and J. Zhang. The application of adaboost for distributed, scalable and on-line learning. In *Proceedings of the fifth ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 362–366. ACM, 1999.

[33] J. Ferber. *Multi-agent systems: an introduction to distributed artificial intelligence*. Addison-Wesley Longman Publishing Co., Inc., 1999.

[34] A. Frank and A. Asuncion. UCI machine learning repository, 2010.

[35] Yoav Freund and Robert E. Schapire. Experiments with a new boosting algorithm. In *Thirteenth International Conference on Machine Learning*, pages 148–156, San Francisco, 1996. Morgan Kaufmann.

[36] J. Friedman, T. Hastie, and R. Tibshirani. Additive logistic regression: a statistical view of boosting (with discussion and a rejoinder by the authors). *The annals of statistics*, 28(2):337–407, 2000.

[37] Q. Fu, S.X. Hu, and S.Y. Zhao. Clustering-based selective neural network ensemble. *Journal of Zhejiang University-Science A*, 6(5):387–392, 2005.

[38] Giorgio Giacinto, Fabio Roli, and Giorgio Fumera. Design of effective multiple classifier systems by clustering of classifiers. In *Proc. of ICPR2000, 15th Int. Conference on Pattern Recognition*, pages 3–8, 2000.

[39] P.R. Graça and G. Gaspar. Using cognition and learning to improve agents' reactions. In *Adaptive agents and multi-agent systems*, pages 239–259. Springer-Verlag, 2003.

[40] Yike Guo and Janjao Sutiwaraphun. Probing knowledge in distributed data mining. In *Proceedings of the Third Pacific-Asia Conference on Methodologies for Knowledge Discovery and Data Mining*, PAKDD '99, pages 443–452, London, UK, UK, 1999. Springer-Verlag.

[41] Z. Gyöngyi, H. Garcia-Molina, and J. Pedersen. Combating web spam with trustrank. In *Proceedings of the Thirtieth international conference on Very large data bases-Volume 30*, pages 576–587. VLDB Endowment, 2004.

[42] Lawrence O. Hall, Nitesh Chawla, and Kevin W. Bowyer. Combining decision trees learned in parallel. In *In Working Notes of the KDD-97 Workshop on Distributed Data Mining*, pages 10–15, 1998.

[43] Lawrence O. Hall, Nitesh Chawla, and Kevin W. Bowyer. Decision tree learning on very large data sets. In *In IEEE Conference on Systems, Man and Cybernetics*, pages 2579–2584, 1998.

[44] M. Hall, E. Frank, G. Holmes, B. Pfahringer, P. Reutemann, and I.H. Witten. The weka data mining software: an update. *ACM SIGKDD Explorations Newsletter*, 11(1):10–18, 2009.

[45] Tin Kam Ho. The random subspace method for constructing decision forests. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 20(8):832–844, 1998.

[46] J. Hübner, L. Vercouter, and O. Boissier. Instrumenting multi-agent organisations with artifacts to support reputation processes. *Coordination, Organizations, Institutions and Norms in Agent Systems IV*, pages 96–110, 2009.

[47] Michael N. Huhns and Gerhard Weiss. Guest editorial. *Machine Learning*, 33(2-3):123–128, 1998.

[48] H. Kargupta, D.H. Byung-Hoon, and E. Johnson. Collective data mining: A new perspective toward distributed data analysis. In *Advances in Distributed and Parallel Knowledge Discovery*, 1999.

[49] N. Kavantzas, D. Burdett, G. Ritzinger, T. Fletcher, Y. Lafon, and C. Barreto. Web services choreography description language version 1.0. *W3C Working Draft*, 17:10–20041217, 2004.

[50] Josef Kittler, Mohamad Hatef, Robert P. W. Duin, and Jiri Matas. On combining classifiers. *IEEE Transactions on pattern analysis and machine intelligence*, 20:226–239, 1998.

[51] J.M. Kleinberg. Authoritative sources in a hyperlinked environment. *Journal of the ACM (JACM)*, 46(5):604–632, 1999.

[52] R. Kota, N. Gibbins, and N.R. Jennings. Self-organising agent organisations. In *Proceedings of The 8th International Conference on Autonomous Agents and Multiagent Systems-Volume 2*, pages 797–804. International Foundation for Autonomous Agents and Multiagent Systems, 2009.

[53] A. Krogh and J. Vedelsby. Neural network ensembles, cross validation, and active learning. *Advances in neural information processing systems*, pages 231–238, 1995.

[54] Ludmila I. Kuncheva. *Combining Pattern Classifiers: Methods and Algorithms*. Wiley-Interscience, 2004.

[55] Aleksandar Lazarevic and Zoran Obradovic. The distributed boosting algorithm. In *Seventh ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, pages 311–316. ACM Press, 2001.

[56] Aleksandar Lazarevic and Zoran Obradovic. Effective pruning of neural network classifier ensembles. In *Neural Networks, 2001. Proceedings. IJCNN'01. International Joint Conference on*, volume 2, pages 796–801. IEEE, 2001.

[57] Siu-wai Leung, Xueping Quan, Paolo Besana, Qian Li, Mark Collins, Dietlind Gerloff, and Dave Robertson. Openknowledge for peer-to-peer experimentation in protein identification by ms/ms. *Automated Experimentation*, 3(1):3, 2011.

[58] Milde M. S. Lira, Ronaldo R. B. de Aquino, Aida A. Ferreira, Manoel A. Carvalho, Otoni Nóbrega Neto, and Gabriela S. M. Santos. Combining multiple artificial neural networks using random committee to decide upon electrical disturbance classification. In *IJCNN*, pages 2863–2868, 2007.

[59] D.D. Margineantu and T.G. Dietterich. Pruning adaptive boosting. In *Machine Learning-International Workshop then Conference*, pages 211–218. Morgan Kaufmann Publishers Inc., 1997.

[60] D. Martin, M. Burstein, J. Hobbs, O. Lassila, D. McDermott, S. McIlraith, S. Narayanan, M. Paolucci, B. Parsia, T. Payne, et al. Owl-s: Semantic markup for web services. *W3C Member submission*, 22:2007–04, 2004.

[61] Gonzalo Martínez-Muñoz and Alberto Suárez. Aggregation ordering in bagging. In *Proc. of the IASTED International Conference on Artificial Intelligence and Applications*, pages 258–263. Citeseer, 2004.

[62] Gonzalo Martínez-Muñoz and Alberto Suárez. Pruning in ordered bagging ensembles. In *Proceedings of the 23rd international conference on Machine learning*, ICML '06, pages 609–616, New York, NY, USA, 2006. ACM.

[63] P. Melville and R.J. Mooney. Constructing diverse classifier ensembles using artificial training examples. In *International Joint Conference on Artificial Intelligence*, volume 18, pages 505–512. DTIC Document, 2003.

[64] M. Minsky. *The society of mind*. Simon and Schuster, 1988.

[65] Thomas M. Mitchell. *Machine Learning*. McGraw-Hill, Inc., New York, NY, USA, 1 edition, 1997.

[66] P.J. Modi and W.M. Shen. Collaborative multiagent learning for classification tasks. In *Proceedings of the fifth international conference on Autonomous agents*, pages 37–38. ACM, 2001.

[67] M.E.J. Newman. Power laws, Pareto distributions and Zipf's law. *Contemporary physics*, 46(5):323–351, 2005.

[68] L. Nunes and E. Oliveira. Cooperative learning using advice exchange. *Adaptive agents and multi-agent systems*, pages 560–560, 2003.

[69] S. Ontañón and E. Plaza. A bartering approach to improve multiagent learning. In *Proceedings of the first international joint conference on Autonomous agents and multiagent systems: part 1*, pages 386–393. ACM, 2002.

[70] David Opitz and Richard Maclin. Popular ensemble methods: An empirical study. *Journal of Artificial Intelligence Research*, 11:169–198, 1999.

[71] N.C. Oza and K. Tumer. Classifier ensembles: Select real-world applications. *Information Fusion*, 9(1):4–20, 2008.

[72] I. Partalas, G. Tsoumakas, I. Katakis, and I. Vlahavas. Ensemble pruning using reinforcement learning. *Advances in Artificial Intelligence*, pages 301–310, 2006.

[73] J. Pasley. How bpel and soa are changing web services development. *Internet Computing, IEEE*, 9(3):60–67, 2005.

[74] C. Peltz. Web services orchestration and choreography. *Computer*, 36(10):46–52, 2003.

[75] D. Pop, V. Negru, and C. Sandru. Multi-agent architecture for knowledge discovery. In *Symbolic and Numeric Algorithms for Scientific Computing, 2006. SYNASC'06. Eighth International Symposium on*, pages 217–226. IEEE, 2006.

[76] M.V.N. Prasad, S.E. Lander, and V.R. Lesser. Cooperative learning over composite search spaces: Experiences with a multi-agent design system. In *Thirteenth National Conference on Artificial Intelligence (AAAI-96)*, 1996.

[77] Dave Robertson. Multi-agent coordination as distributed logic programming. *Logic programming*, pages 77–96, 2004.

[78] Dave Robertson. A lightweight coordination calculus for agent systems. *Declarative agent languages and technologies II*, pages 109–115, 2005.

[79] Dave Robertson. Webs of interactions: Exploring peer ranking via simulation in openknowledge. Technical report, School of Informatics, The University of Edinburgh, 2009.

[80] Dave Robertson et al. Open knowledge: Semantic webs through peer-to-peer interaction. openknowledge manifesto. Technical report, Information Engineering and Computer Science, The University of Trento, 2006.

[81] J.J. Rodriguez, L.I. Kuncheva, and C.J. Alonso. Rotation forest: A new classifier ensemble method. *Pattern Analysis and Machine Intelligence, IEEE Transactions on*, 28(10):1619–1630, 2006.

[82] L. Rokach. Ensemble-based classifiers. *Artificial Intelligence Review*, 33(1):1–39, 2010.

[83] L. Rokach. *Pattern classification using ensemble methods*, volume 75. World Scientific Pub Co Inc, 2010.

[84] J. Sabater, M. Paolucci, and R. Conte. Repage: Reputation and image among limited autonomous partners. *Journal of Artificial Societies and Social Simulation*, 9(2), 2006.

[85] J. Sabater and C. Sierra. Regret: reputation in gregarious societies. In *Proceedings of the fifth international conference on Autonomous agents*, pages 194–195. ACM, 2001.

[86] Amanda Schierz. Virtual screening of bioassay data. *Journal of Cheminformatics*, 1(1):21, 2009.

[87] R.M. Shapiro. Xpdl 2.0: Integrating process interchange and bpmn. *Workflow Handbook*, pages 183–194, 2006.

[88] Carles Sierra and John Debenham. An information-based model for trust. In *Proceedings of the fourth international joint conference on Autonomous agents and multiagent systems*, AAMAS '05, pages 497–504, New York, NY, USA, 2005. ACM.

[89] S. Stolfo, A.L. Prodromidis, S. Tselepis, W. Lee, D.W. Fan, and P.K. Chan. Jam: Java agents for meta-learning over distributed databases. In *Proceedings of the 3rd International Conference on Knowledge Discovery and Data Mining*, pages 74–81, 1997.

[90] D. Szer and F. Charpillet. Coordination through mutual notification in cooperative multiagent reinforcement learning. In *Proceedings of the Third International Joint Conference on Autonomous Agents and Multiagent Systems-Volume 3*, pages 1254–1255. IEEE Computer Society, 2004.

[91] M. Tan. Multi-agent reinforcement learning: Independent vs. cooperative agents. In *Proceedings of the tenth international conference on machine learning*, volume 337. Amherst, MA, 1993.

[92] Grigorios Tsoumakas, Lefteris Angelis, and Ioannis Vlahavas. Selective fusion of heterogeneous classifiers. *Intell. Data Anal.*, 9(6):511–525, November 2005.

[93] Grigorios Tsoumakas, Ioannis Katakis, and Ioannis Vlahavas. Effective voting of heterogeneous classifiers. In *In Proceedings of the 15th European Conference on Machine Learning*, pages 465–476, 2004.

[94] A. Turinsky, R. Grossman, et al. A framework for finding distributed data mining strategies that are intermediate between centeralized strategies and in-place strategies. In *Proceedings of Workshop on Distributed and Parallel Knowledge Discovery at KDD-2000*, pages 1–7, 2000.

[95] W.M.P. van der Aalst and A. H. M. Ter Hofstede. Yawl: Yet another workflow language. *Information Systems*, 30:245–275, 2003.

[96] R. Vilalta, C. Giraud-Carrier, P. Brazdil, and C. Soares. Using meta-learning to support data mining. *International Journal of Computer Science and Applications*, 1(1):31–45, 2004.

[97] T. Vu, R. Powers, and Y. Shoham. Learning against multiple opponents. In *Proceedings of the fifth international joint conference on Autonomous agents and multiagent systems*, pages 752–759. ACM, 2006.

[98] W.P. Walters, M.T. Stahl, and M.A. Murcko. Virtual screening-an overview. *Drug Discovery Today*, 3(4):160–178, 1998.

[99] Z.G. Wang, X.H. Liang, and Q.P. Zhao. Adaptive mechanisms of organizational structures in multi-agent systems. *Agent Computing and Multi-Agent Systems*, pages 471–477, 2006.

[100] Gerhard Weiß. Learning to coordinate actions in multi-agent systems. In *in Proceedings of the 13th International Conference on Artificial Intelligence*, pages 311–316. Morgan Kaufmann, 1993.

[101] Gerhard Weiß and Pierre Dillenbourg. *What is 'multi' in multi-agent learning?*, pages 64–80. Pergamon Press, Oxford, 1999.

[102] S.D. Whitehead and D.H. Ballard. *A study of cooperative mechanisms for faster reinforcement learning*. University of Rochester, Department of Computer Science, 1991.

[103] David H. Wolpert. Stacked generalization. *Neural Networks*, 5:241–259, 1992.

[104] Q. Yang and X. Wu. 10 challenging problems in data mining research. *International Journal of Information Technology & Decision Making*, 5(4):597–604, 2006.

[105] G. Zacharia, A. Moukas, and P. Maes. Collaborative reputation mechanisms for electronic marketplaces. *Decision Support Systems*, 29(4):371–388, 2000.

[106] Yi Zhang, Samuel Burer, and W. Nick Street. Ensemble pruning via semi-definite programming. *J. Mach. Learn. Res.*, 7:1315–1338, December 2006.

[107] Z.H. Zhou and W. Tang. Selective ensemble of decision trees. *Rough Sets, Fuzzy Sets, Data Mining, and Granular Computing*, pages 589–589, 2003.